STAT529
Applied Bayesian Decision Theory
Homework 3

Fangda Li
li1208@purdue.edu

April 13, 2017

1 Lamb Diets

Problem setup: $X_{ij} \sim N(\theta_i, \sigma_i^2)$, $\theta_i \sim N(\beta, \tau^2)$, $\beta \sim N(\beta_0, A^2) \sim h_1(\cdot)$, $\tau^2 \sim inv\Gamma(\alpha_2, \beta_2) \sim h_2(\cdot)$ and $\sigma_i^2 \sim inv\Gamma(\alpha_3, \beta_3) \sim h_3(\cdot)$ for $i = 1, 2, ..., 5$ and $j = 1, 2, ..., N_i$. In this problem, we are particularly interested in generating samples from the posterior distribution of θ_i. Since we don't know the form of the posteriors well enough, Gibbs sampler is used to generate θ_i samples from the conditional distribution $g(\theta_i|rest)$. After a large number of iterations, the samples generated from the conditional distribution are shown to represent the desired posteriors. Assuming all the parameters are statistically independent, the overarching conditional distribution can be shown to be as follows:

$$g(z|rest) = \frac{\prod_{i=1}^{5}\prod_{j=1}^{N_i} f(x_{ij}|\theta_i)\pi(\theta_i|\beta,\tau^2)h_1(\beta)h_2(\tau^2)h_3(\sigma^2)}{\int \text{above } dz},$$

where z is the parameter of interest. For example, taking θ_1 as our parameter of interest, we can simply the conditional distribution to:

$$\begin{aligned} g(\theta_1|rest) &= \frac{e^{-\frac{(\theta_1-\bar{x}_1)^2}{2\frac{\sigma_1^2}{n_1}}} e^{-\frac{(\theta_1-\beta)^2}{2\tau^2}}}{\int \text{above } d\theta_1} \\ &\sim N(m_1, v_1^2), \end{aligned}$$

where $m1 = \frac{\frac{\sigma_1^2}{n_1}\beta+\tau^2\bar{x}_1}{\frac{\sigma_1^2}{n_1}+\tau^2}$, $v_1^2 = \frac{\frac{\sigma_1^2}{n_1}\tau^2}{\frac{\sigma_1^2}{n_1}+\tau^2}$, and n_1 is the number of samples in the data for diet 1. Similarly, we can show the conditional probabilities for the rest of parameters:

$$g(\sigma_1^2|rest) \sim inv\Gamma(\alpha_3 + \frac{n_1}{2}, \frac{2\beta_3}{2+\beta_3 n_1[S_1^2 + (\bar{X}_1-\theta_1)^2]}),$$

$$g(\beta|rest) \sim N(\frac{\frac{\tau^2}{5}\beta_0 + A^2\bar{\theta}}{\frac{\tau^2}{5}+A^2}, \frac{\frac{\tau^2}{5}A^2}{\frac{\tau^2}{5}+A^2}),$$

$$g(\tau^2|rest) \sim inv\Gamma(\alpha_2 + \frac{5}{2}, \frac{2\beta_2}{2+\beta_2\sum(\theta_i-\beta)^2}).$$

Using the conditional distributions above, we are able to generate many sequences of parameters sequentially, each of which is based on its preceding sequence. Finally, the samples from the generated sequences will represent the posterior distributions of interest, as shown in Figure 1.

a) **Method I**: In order to find the Bayes selected diet, we compute the expected posterior loss (EPL) using the zero-one-loss function and the posterior:

$$\begin{aligned} E[L(a_i,\theta)|X] &= \sum_{\theta} L(a_i,\theta)\Pi(\theta|X) \\ &= \sum_{\theta\neq\theta_{[5]}} L(a_i,\theta)\Pi(\theta|X) \\ &= 1 - P(\{\theta_i = \theta_{[5]}\}|X). \end{aligned}$$

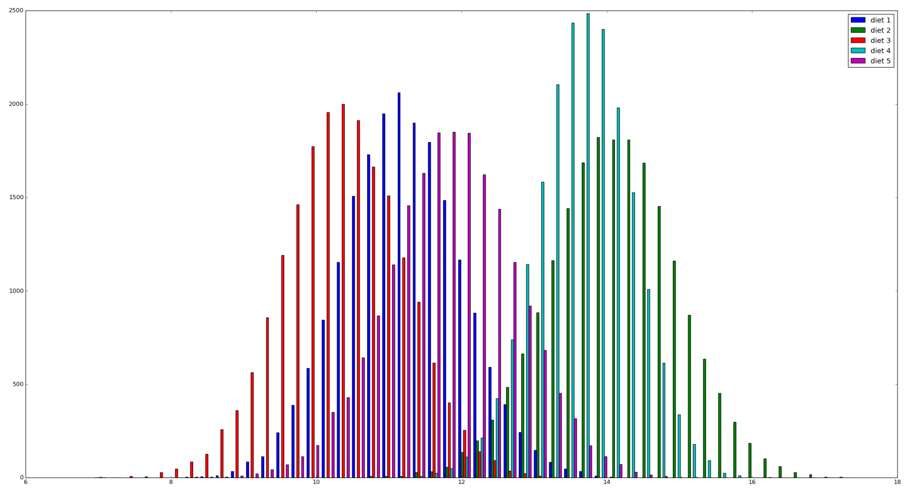

Figure 1: Posterior distributions of all five diets.

According to my simulation, the EPL for the five diets are $0.998, 0.341, 1.000, 0.670, 0.991$, respectively. As a result, the Bayes selected diet is **Diet 2**.

Method II: In order to find the Bayes subsets of diets, we first define the following loss function for the chosen subset S_j:

$$L(S_j, \theta) = \begin{cases} \frac{|S_j|}{5}, \text{if } \theta_{[5]} \in S_j, \\ 1 + \frac{|S_j|}{5}, \text{if } \theta_{[5]} \notin S_j. \end{cases}$$

Now, we reorganize the order of diets in a descending order of the posterior probabilities such that $P_1 \geq P_2 \geq P_3 \geq P_4 \geq P_5$. Then, the EPL for a chosen subset S_j becomes:

$$\begin{aligned} E[L(a_i, \theta)|X] &= \sum_{\theta} L(a_i, \theta)\Pi(\theta|X) \\ &= \frac{|S_j|}{5} P(\{\theta_{[5]} \in S_j\}) + (1 + \frac{|S_j|}{5}) P(\{\theta_{[5]} \notin S_j\}) \\ &= \frac{|S_j|}{5} P(\{\theta_{[5]} \in S_j\}) + (1 + \frac{|S_j|}{5})(1 - P(\{\theta_{[5]} \in S_j\})) \\ &= 1 + \frac{|S_j|}{5} - P(\{\theta_{[5]} \in S_j\}) \\ &= 1 + \frac{|S_j|}{5} - \sum_{i=1}^{|S_j|} P_i. \end{aligned}$$

As a result of EPL, we keep adding diet i to S_j until $P_i > \frac{1}{5}$. According to my simulation, the posterior probabilities P_i are $0.002, 0.660, 0.000, 0.330, 0.008$, respectively. Therefore, the Bayes selected subset contains two diets: **{Diet 2, Diet 4}**.

b) Given the generated samples of the posterior distribution, we simply use counting to obtain the following results:

$$\begin{aligned} P(\theta_2 \geq \theta_3) &= 0.999, \\ P(\theta_2 \geq \theta_3 + 2) &= 0.931, \\ P(\theta_2 \geq \theta_3 + 4) &= 0.421, \\ P(\theta_2 \geq \theta_3 + 6) &= 0.032. \end{aligned}$$

c) Similar to the previous step, we use counting again to obtain the following results:

$$\begin{aligned} P(\theta_2 \geq 14 | \theta_3 \leq 10) &= 0.550, \\ P(\theta_2 \geq 14) &= 0.552. \end{aligned}$$

Since $P(\theta_2 \geq 14|\theta_3 \leq 10) \approx P(\theta_2 \geq 14)$, the two events are statistically independent.

2 Corn Data

In this problem, we aim to build a Hierarchical Bayes model and generate the posterior distributions using the Markov Chain Monte Carlo method. The HB model is setup as follows: $\beta \sim h_1(\cdot) \sim N(125, 15^2), \tau^2 \sim h_2(\cdot) \sim uniform(2, 16), \theta_i \sim N(\beta, \tau^2)$, where $i = 1 \ldots 12$ represents each of the cells in the data.

In order to find the best species, for each species of corn $j = 1 \ldots 4$, we have $\pi(\lambda_j|\beta, \tau^2) \sim N(\beta, \frac{\tau^2}{3}), x_j \sim N(\theta_j, \sigma^2) \sim N(\theta_j, 80), f(\overline{Y}_j|\lambda_j) \sim N(\lambda_j, \frac{\sigma^2}{12})$, where, for example, $\lambda_1 = \frac{\theta_1+\theta_2+\theta_3}{3}$ and $\overline{Y}_1 = \frac{\bar{x}_1+\bar{x}_2+\bar{x}_3}{3}$. Now, using the setup above, we can generate $\vec{\lambda}^*$ at time t and keep it with probability r, which is:

$$r = min\{\frac{\prod_j f(\overline{Y}_j|\vec{\lambda}^*, \beta^*, \tau^{2*})}{\prod_j f(\overline{Y}_j|\vec{\lambda}^{t-1}, \beta^{t-1}, \tau^{2t-1})}, 1\}.$$

After obtaining a sufficient number of keepers (posterior samples) for $\vec{\lambda}$, as shown in Figure 2, we can simply using the counting technique to compute the probabilities of interest. According to simulation, the following result is obtained:

$$P(\lambda_2 = \lambda_{[4]}|\text{data}) = 0.533.$$

Using an identical approach for the fertilizers, the posterior distribution for $\vec{W}$ is shown in Figure 3 and the following result is obtained:

$$P(W_1 = W_{[3]}|\text{data}) = 0.713.$$

Again, for θ_4 and θ_{12}, the posterior distribution is shown in Figure 4 and the following result is obtained:

$$P(\theta_{12} \geq \theta_4|\text{data}) = 0.975.$$

Although λ_2 and W_1 both have the highest probability to be the best, their intersection, θ_4, does not have a high probability to be the best among all the possible combinations, because the probability of θ_{12} being better than θ_4 is almost equal to 1.

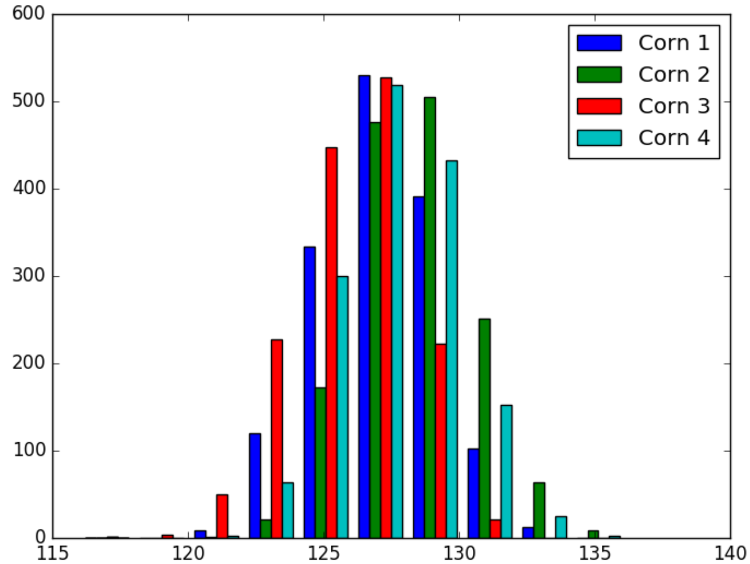

Figure 2: Posterior distributions of all four corn species.

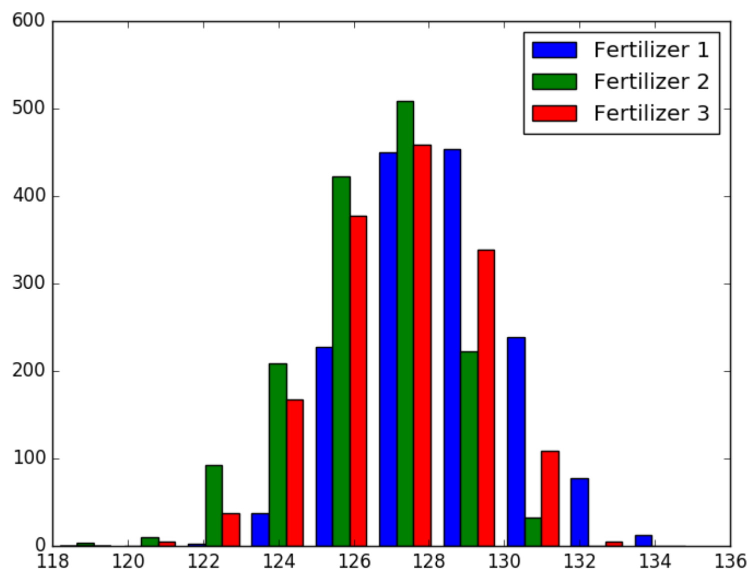

Figure 3: Posterior distributions of all three corn fertilizers.

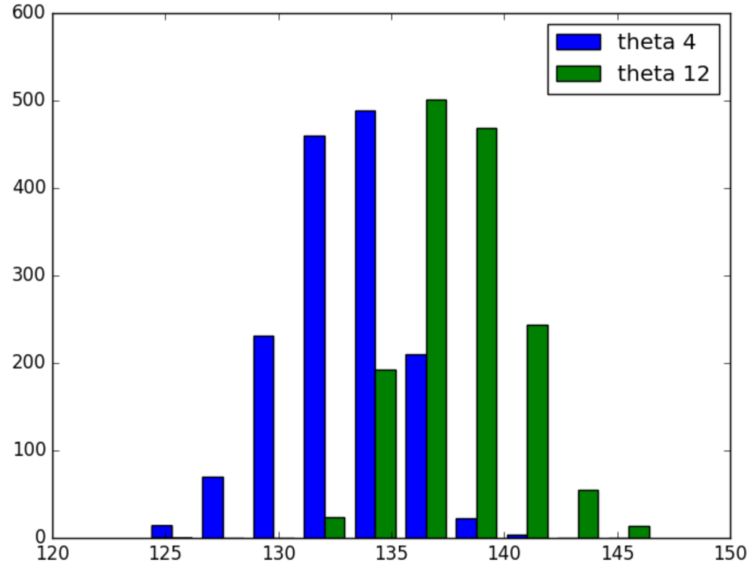

Figure 4: Posterior distributions of θ_4 and θ_{12}.

3 Air Conditioner

In this problem, we aim to build a Hierarchical Bayes model and generate the posterior distributions using the Markov Chain Monte Carlo method. Let θ_i denote the true mean time of failure of air conditioner i, and $x_{i,j}$ denote the jth observed time of failure of air conditioner i. Now, using the HB model with $h_1(\alpha) \sim uniform(30, 65)$ and $h_2(\beta|\alpha) = uniform(-0.057\alpha + 5, -0.057\alpha + 6.5)$, we can generate θ_i^* from a Gamma distribution with parameters α and β:

$$\pi(\theta|\alpha, \beta) = \frac{1}{\Gamma(\alpha)\beta^\alpha}\theta^{\alpha-1}e^{-\frac{\theta}{\beta}}.$$

Since $x_{i,j}$ is Poisson distributed, the likelihood function of $\vec{x_i}$ is:

$$f(\vec{x_i}|\theta_i) = (\frac{1}{\theta_i})^{n_i}e^{-\frac{n_i\bar{x_i}}{\theta_i}},$$

where n_i is the number of observed failures for air conditioner i. Finally, the MCMC process is formulated as follows: generate θ_i^* from $\pi(\theta|\alpha, \beta)$ and keep with probability r, which is defined as:

$$r = min\{\frac{\prod_i f(\vec{x_i}|\theta_i^*)}{\prod_i f(\vec{x_i}|\theta_i^{t-1})}, 1\}.$$

In this problem, since we are only concerned with the relationship between air conditioner 1 and 2, r further becomes:

$$\begin{aligned} r &= min\{\frac{f(\vec{x_1}|\theta_1^*)f(\vec{x_2}|\theta_2^*)}{f(\vec{x_1}|\theta_1^{t-1})f(\vec{x_2}|\theta_2^{t-1})}, 1\} \\ &= min\{(\frac{\theta_1^{t-1}}{\theta_1^*})^{14}e^{-14\bar{x_1}(\frac{1}{\theta_1^*}+\frac{1}{\theta_1^{t-1}})}(\frac{\theta_2^{t-1}}{\theta_2^*})^{14}e^{-14\bar{x_2}(\frac{1}{\theta_2^*}+\frac{1}{\theta_2^{t-1}})}, 1\}. \end{aligned}$$

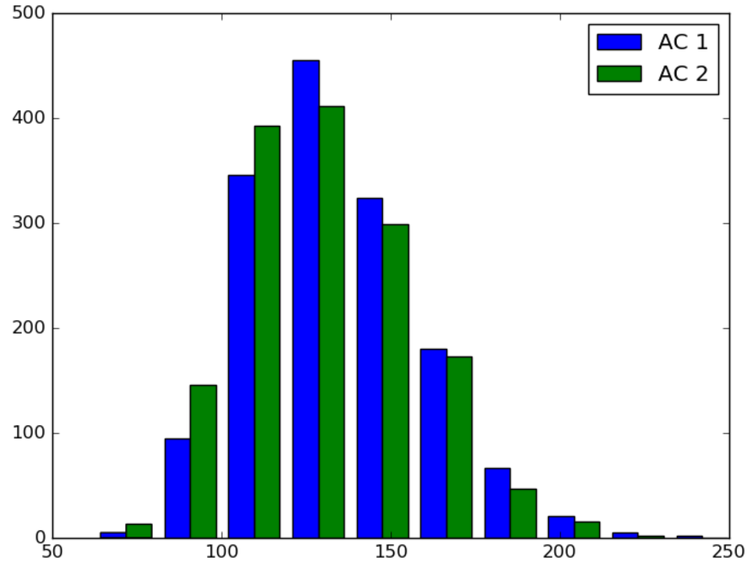

Figure 5: Posterior distributions of θ_1 and θ_2.

1. The posterior distribution generated in my simulation using HB is shown in Figure 5 and the following posterior probabilities are obtained:

$$P(\theta_1 \geq \theta_2 + 10) = 0.389,$$

$$P(\theta_1 \geq \theta_2 + 20) = 0.257.$$

2. The posterior distribution generated in my simulation using static α/β is shown in Figure 6 and the following posterior probabilities are obtained:

$$P(\theta_1 \geq \theta_2 + 10) = 0.386,$$

$$P(\theta_1 \geq \theta_2 + 20) = 0.235.$$

3. The results are nearly identical.

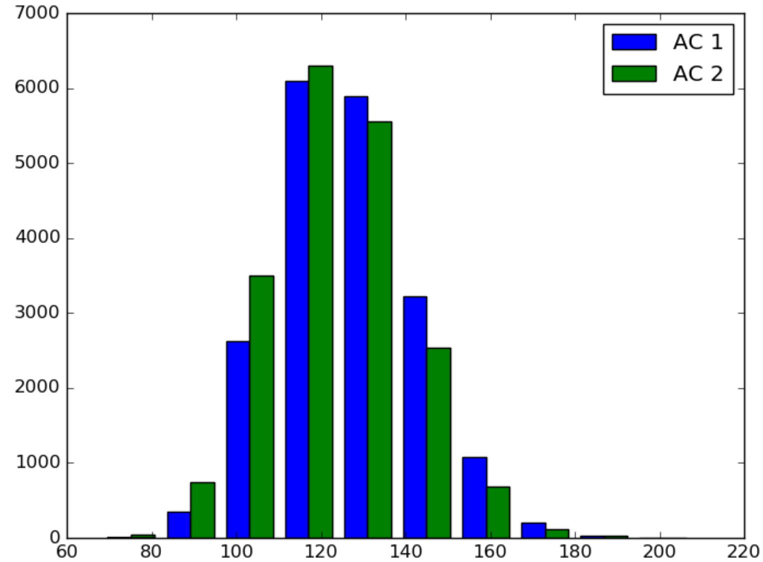

Figure 6: Posterior distributions of θ_1 and θ_2.

4 Lamb Diets 2

In this problem, we are interested to find the predictive probability $P(y_2 > y_3 + b)$. Given parameters are $h_1(\beta) \sim N(12, 2^2)$, $h_2(\tau^2) \sim \Gamma(2.5, 2)$, $\sigma_2^2 = \sigma_3^2 = 4$ and $n_2 = n_3 = 8$. In order to calculate the predictive probability, we need to first obtain the predictive distribution $f(y_2, y_3|\bar{x}_1, \bar{x}_2)$. Recall from notes:

$$\begin{aligned} f(y_2, y_3|\bar{x}_1, \bar{x}_2) &= \frac{\int\int f(y_2, y_3, \bar{x}_2, \bar{x}_3|\theta_2, \theta_3)\pi(\theta_2, \theta_3|\beta, \tau^2)d\beta d\tau^2}{f(\bar{x}_2, \bar{x}_3)} \\ &= \int\int \tilde{f}(y_2)\tilde{f}(y_3)h(\beta, \tau^2|\bar{x}_2, \bar{x}_3)d\beta d\tau^2, \end{aligned}$$

where $\tilde{f}(y_2) \sim N(m_2, \sigma_2^2 + v_2^2)$, $m_2 = \frac{\frac{\sigma_2^2}{n_2}\beta + \tau^2\bar{x}_2}{\frac{\sigma_2^2}{n_2} + \tau^2}$, and $v_2^2 = \frac{\frac{\sigma_2^2}{n_2}\tau^2}{\frac{\sigma_2^2}{n_2} + \tau^2}$. The distribution $\tilde{f}(y_3)$ and its parameters have the same form.

Then, we use MCMC to approximate the posterior distribution $h(\beta, \tau^2|\bar{x}_1, \bar{x}_2)$. The MCMC is formulated as follows: (1) generate β^*, τ^{2*} samples from $h_1(\beta)$ and $h_2(\tau^2)$; (2) At time t, keep the β^*, τ^{2*} samples with probability r:

$$r = min\{\frac{f(\bar{x}_2|\beta^*, \tau^{2*})f(\bar{x}_3|\beta^*, \tau^{2*})}{f(\bar{x}_2|\beta^{t-1}, \tau^{2t-1})f(\bar{x}_3|\beta^{t-1}, \tau^{2t-1})}, 1\}.$$

Finally, the predictive probability can be written as:

$$P(y_2 - y_3 > 6|\bar{x}_2, \bar{x}_3) = \int\int_{y_2<y_3+b}\int\int \tilde{f}(y_2)\tilde{f}(y_3)h(\beta, \tau^2|\bar{x}_2, \bar{x}_3)d\beta d\tau^2 dy_2 dy_3$$
$$= \int\int \Phi(\frac{m_2 - m_3 - b}{\sqrt{\sigma_2^2 + \sigma_3^2 + v_2^2 + v_3^2}})h(\beta, \tau^2|\bar{x}_2, \bar{x}_3)d\beta d\tau^2.$$

For each generated β and τ^2 sample pair, we can calculate its corresponding probability $\Phi(\cdot)$ and use the average as the final predictive probability. According to my simulation, the following results are obtained:

$$P(y_2 > y_3 + 0) = 0.924,$$

$$P(y_2 > y_3 + 1) = 0.864,$$

$$P(y_2 > y_3 + 3) = 0.668,$$

$$P(y_2 > y_3 + 5) = 0.409.$$

5 Code

5.1 Q1

```
from pylab import *

def get_gamma_pdf(x, alpha, beta):
  '''
    Return the pdf of custom gamma function on given points.
  '''
  return 1.0 / (beta**alpha * gamma(alpha)) * x**(alpha+1) * exp(-x/beta)

def invgamma_rvs(alpha, beta, N=1):
  '''
    Return samples of the custom inverse Gamma RV.
  '''
  r = arange(0,20.01,0.01)
  gamma_pdf = get_gamma_pdf(r, alpha, beta)
  gamma_pdf = gamma_pdf / sum(gamma_pdf)
  if N == 1:
    return 1. / choice(r, p=gamma_pdf)
  else:
    return 1. / choice(r, size=N, p=gamma_pdf)

def get_posterior_samples_gibbs(X):
  '''
    Use Gibbs sample to generate posterior samples for A3Q1.
```

```
    X is a list of arrays for each diet.
  '''
  N = 20000
  # Some preprocessing
  X_num = array([1.0*len(X[i]) for i in range(5)])
  X_mean = array([mean(X[i]) for i in range(5)])
  # Initial guesses
  beta = 2*randn() + 12
  tau2 = invgamma_rvs(6, 0.05)
  sigma2_i = invgamma_rvs(13.1, 0.0083)
  # Posteriors, each column represents one diet
  theta_ij = zeros((N,5))
  theta_ij[0] = sqrt(tau2)*randn(5) + beta
  # Generating loop
  for j in range(1,N):
    theta_i = theta_ij[j-1]
    beta = sqrt(tau2*4/(tau2+20))*randn() + (tau2*12+5*4*mean(theta_i)) / (tau2+5*4)
    tau2 = invgamma_rvs(2.5 + 6, 2*0.05 / (2 + 0.05*sum((theta_i-beta)**2)))
    sigma2_i = array([invgamma_rvs(13.1+X_num[i]/2., \
        2 / ( 2/0.0083 + sum( (X[i] - X_mean[i])**2 ) + X_num[i]*( X_mean[i] - theta_i[i] ) )) \
        for i in range(5)])
    theta_i = array([sqrt( sigma2_i[i]*tau2 / (sigma2_i[i] + X_num[i]*tau2) ) * randn() \
        + (sigma2_i[i]*beta + tau2*X_num[i]*X_mean[i]) / (sigma2_i[i] + X_num[i]*tau2) \
        for i in range(5)])
    theta_ij[j] = theta_i
  # Discard the first 500 to minimize the influence of the intial guess
  theta_ij = theta_ij[500:]
  figure()
  hist(theta_ij, bins=50, label=["diet %d"%(i) for i in range(1,6)])
  legend()
  show()
  return theta_ij

def main():
  set_printoptions(precision=3)
  X = [array([12.7,6.6,14.7,12.2,4.4,7.8,13.8,13.7,11.1,9.1,14.0]),
    array([17.1,11.9,12.7,16.8,15.0,14.6,13.7,16.4]),
    array([5.2,4.5,10.5,15.0,5.0,14.9,7.6,8.3,10.8,14.6,15.1,7.0,9.3]),
    array([14.3,16.2,10.0,13.1,16.9,11.2,10.1,18.3,13.5,15.0,15.1,14.8,15.7,13.2,12.2,13
    array([10.5,7.5,4.7,12.5,13.1,13.5,12.2,16.1,9.0,17.9])]
  posteriors = get_posterior_samples_gibbs(X)
```

```
  # Q1a
  # Find index of max theta for every row
  max_indices = argmax(posteriors, axis=1)
  indices, counts = unique(max_indices, return_counts=True)
  epl, P = ones(5), zeros(5)
  epl[indices] = 1 - 1.0 * counts / sum(counts)
  P[indices] = 1.0 * counts / sum(counts)
  print "EPL:", epl
  print "Posterior P:", P
  print "Bayes selected diet:", argmin(epl)+1
  # Q1b
  P_b = zeros(4)
  for i, b in enumerate([0,2,4,6]):
    P_b[i] = sum(posteriors[:,1] >= posteriors[:,2] + b) * 1.0 /
     ↪ posteriors.shape[0]
  print "P_b:", P_b
  # Q1c
  P_2 = sum(posteriors[:,1] >= 14) * 1.0 / posteriors.shape[0]
  temp = posteriors[:,1][posteriors[:,2] <= 10] # P(theta2 | theta3 <= 10)
  P_23 = sum(temp >= 14) * 1.0 / temp.shape[0]
  print "P(theta2 >= 14 | theta3 <= 10):", P_23
  print "P(theta2 >= 14):", P_2

if __name__ == '__main__':
  main()
```

5.2 Q2

```
from pylab import *

def get_corn_posterior_samples_MCMC(X):
  '''
    Using a HB model, A/R algorithm with MCMC to generate posterior
 ↪  samples.
  '''
  beta0 = 125.
  A = 15.
  sigma2 = 80.
  N = 2000
  # Some preprocessing
  Y_mean = mean(X.reshape(4,-1), axis=1)
  # Posteriors before selection, each column represents one species
  lambda_ij = zeros((N, 4))
  # Initial guesses
  beta = randn()*A + beta0
```

```
    tau2 = rand()*14 + 2
    lambda_ij[0] = randn(4)*sqrt(tau2/3) + beta
    # Generating loop
    j = 1
    while j < N:
      beta = randn()*A + beta0
      tau2 = rand()*14 + 2
      lambda_i = randn(4)*sqrt(tau2/3) + beta
      # Determine whether to keep
      r = min(exp( -0.5 * sum((lambda_i - Y_mean)**2 - (lambda_ij[j-1] -
       ↪  Y_mean)**2) / (sigma2 / 12) ),1)
      # Keep with probability r
      if rand() > r: continue
      lambda_ij[j] = lambda_i
      j += 1
    # Discard the first 500 to minimize the influence of the intial guess
    lambda_ij = lambda_ij[500:]
    # hist(lambda_ij, bins=10, label=["Corn %d"%(i) for i in range(1,5)])
    # legend()
    # show()
    return lambda_ij

def get_fertilizer_posterior_samples_MCMC(X):
    '''
      Using a HB model, A/R algorithm with MCMC to generate posterior
 ↪    samples.
    '''
    beta0 = 125.
    A = 15.
    sigma2 = 80.
    N = 2000
    # Some preprocessing
    T_mean = mean(X.T, axis=1)
    # Posteriors before selection, each column represents one fertilizer
    w_ij = zeros((N, 3))
    # Initial guesses
    beta = randn()*A + beta0
    tau2 = rand()*14 + 2
    w_ij[0] = randn()*sqrt(tau2/4) + beta
    # Generating loop
    j = 1
    while j < N:
      beta = randn()*A + beta0
      tau2 = rand()*14 + 2
      w_i = randn(3)*sqrt(tau2/4) + beta
```

```
    # Determine whether to keep
    r = min(exp( -0.5 * sum((w_i - T_mean)**2 - (w_ij[j-1] - T_mean)**2) /
     ↪  (sigma2 / 16) ),1)
    # Keep with probability r
    if rand() > r: continue
    w_ij[j] = w_i
    j += 1
  # Discard the first 500 to minimize the influence of the intial guess
  w_ij = w_ij[500:]
  # hist(w_ij, bins=10, label=["Fertilizer %d"%(i) for i in range(1,4)])
  # legend()
  # show()
  return w_ij

def get_combined_posterior_samples_MCMC(X):
  '''
    Using a HB model, A/R algorithm with MCMC to generate posterior
↪   samples.
  '''
  beta0 = 125.
  A = 15.
  sigma2 = 80.
  N = 2000
  # Some preprocessing
  X_mean = mean(X.T.flatten().reshape(-1,4),axis=1)
  X_mean = X_mean[array([1,4,7,10,2,5,8,11,3,6,9,12])-1]
  X_mean = X_mean[[3,11]]
  # Posteriors before selection, each column represents one fertilizer
  theta_ij = zeros((N, 2))
  # Initial guesses
  beta = randn()*A + beta0
  tau2 = rand()*14 + 2
  theta_ij[0] = randn()*sqrt(tau2) + beta
  # Generating loop
  j = 1
  while j < N:
    beta = randn()*A + beta0
    tau2 = rand()*14 + 2
    theta_i = randn(2)*sqrt(tau2) + beta
    # Determine whether to keep
    r = min(exp( -0.5 * sum((theta_i - X_mean)**2 - (theta_ij[j-1] -
     ↪  X_mean)**2) / (sigma2 / 16) ),1)
    # Keep with probability r
    if rand() > r: continue
    theta_ij[j] = theta_i
```

```
    j += 1
  # Discard the first 500 to minimize the influence of the intial guess
  theta_ij = theta_ij[500:]
  hist(theta_ij, bins=10, label=["theta %d"%(i) for i in [4,12] ])
  legend()
  show()
  return theta_ij

def main():
  set_printoptions(precision=3)
  X = array(([138,122,121], [138,127,119], [129,124,118], [131,123,122],
        [135,130,133], [140,140,130], [136,140,132], [130,128,128],
        [125,120,115], [140,115,112], [140,110,110], [125,120,115],
        [130,118,141], [130,115,140], [140,112,139], [125,116,142]))
  # Q2
  # Corn-wise
  lambda_ij = get_corn_posterior_samples_MCMC(X)
  P2c = 1.0*sum(argmax(lambda_ij, axis=1) == 1) / lambda_ij.shape[0]
  print"P(lambda2 | X):", P2c
  # Fertilizer-wise
  w_ij = get_fertilizer_posterior_samples_MCMC(X)
  P1f = 1.0*sum(argmax(w_ij, axis=1) == 0) / w_ij.shape[0]
  print"P(W1 | X):", P1f
  # theta4 vs theta12
  theta_ij = get_combined_posterior_samples_MCMC(X)
  P124 = 1.0*sum(argmax(theta_ij, axis=1) == 1) / theta_ij.shape[0]
  print"P(theta12 >= theta4 | X):", P124

if __name__ == '__main__':
  main()
```

5.3 Q3

```
from pylab import *

def get_gamma_pdf(x, alpha, beta):
  '''
    Return the pdf of custom gamma function on given points.
  '''
  return 1.0 / (beta**alpha * gamma(alpha)) * x**(alpha-1) * exp(-x/beta)

def gamma_rvs(alpha, beta, N=1):
  '''
    Return samples of the custom inverse Gamma RV.
  '''
```

```
  r = arange(0,300.1,0.1)
  gamma_pdf = get_gamma_pdf(r, alpha, beta)
  gamma_pdf = gamma_pdf / sum(gamma_pdf)
  if N == 1:
    return choice(r, p=gamma_pdf)
  else:
    return choice(r, size=N, p=gamma_pdf)

def get_ac_posterior_samples_MCMC_HB(X):
  '''
    Using a HB model, Metropolis Hasting algorithm with MCMC to generate
 ↪  posterior samples.
  '''
  N = 2000
  # Some preprocessing, we are only interested in AC 1 and 2
  X = X[:,:2]
  X_mean = X[1]*1.0 / X[0]
  X_num = X[0]
  # Posteriors before selection, each column represents one AC
  theta_ij = zeros((N, X.shape[1]))
  # Initial guesses
  alpha = rand()*35 + 30
  beta = rand()*1.5 - 0.057*alpha + 5
  theta_ij[0] = gamma_rvs(alpha, beta, N=X.shape[1])
  # Generating loop
  j = 1
  while j < N:
    alpha = rand()*35 + 30
    beta = rand()*1.5 - 0.057*alpha + 5
    theta_i = gamma_rvs(alpha, beta, N=X.shape[1])
    # Determine whether to keep
    r = prod( (theta_ij[j-1]/theta_i)**X_num *
     ↪  exp(X_num*X_mean*(1/theta_ij[j-1] - 1/theta_i)) )
    r = min(r, 1)
    # Keep with probability r
    if rand() > r: continue
    theta_ij[j] = theta_i
    j += 1
  # Discard the first 500 to minimize the influence of the intial guess
  theta_ij = theta_ij[500:]
  figure()
  hist(theta_ij, bins=10, label=["AC %d"%(i) for i in range(1,3)])
  legend()
  show()
  return theta_ij
```

```
def get_ac_posterior_samples_MCMC(X):
  '''
    Using a HB model, Metropolis Hasting algorithm with MCMC to generate
↪  posterior samples.
  '''
  N = 20000
  # Some preprocessing, we are only interested in AC 1 and 2
  X = X[:,:2]
  X_mean = X[1]*1.0 / X[0]
  X_num = X[0]
  # Posteriors before selection, each column represents one AC
  theta_ij = zeros((N, X.shape[1]))
  # Initial guesses
  alpha = 50.
  beta = 2.5
  theta_ij[0] = gamma_rvs(alpha, beta, N=X.shape[1])
  # Generating loop
  j = 1
  while j < N:
    alpha = 50.
    beta = 2.5
    theta_i = gamma_rvs(alpha, beta, N=X.shape[1])
    # Determine whether to keep
    r = prod( (theta_ij[j-1]/theta_i)**X_num *
     ↪  exp(X_num*X_mean*(1/theta_ij[j-1] - 1/theta_i)) )
    r = min(r, 1)
    # Keep with probability r
    if rand() > r: continue
    theta_ij[j] = theta_i
    j += 1
  # Discard the first 500 to minimize the influence of the intial guess
  theta_ij = theta_ij[500:]
  figure()
  hist(theta_ij, bins=10, label=["AC %d"%(i) for i in range(1,3)])
  legend()
  show()
  return theta_ij

def main():
  X = array([[14,12,23,16,27,30],
    [1832,1297,2201,1312,2074,1788]])
  # Q3a
  # We only compute posteriors for AC 1 and 2
  posteriors = get_ac_posterior_samples_MCMC_HB(X)
```

```
  P_b = zeros(2)
  for i, b in enumerate([10,20]):
    P_b[i] = sum(posteriors[:,0] >= posteriors[:,1] + b) * 1.0 /
     ↪  posteriors.shape[0]
  print "P_b:", P_b
  # Q3b
  posteriors = get_ac_posterior_samples_MCMC(X)
  P_b = zeros(2)
  for i, b in enumerate([10,20]):
    P_b[i] = sum(posteriors[:,0] >= posteriors[:,1] + b) * 1.0 /
     ↪  posteriors.shape[0]
  print "P_b:", P_b

if __name__ == '__main__':
  main()
```

5.4 Q4

```
from pylab import *
from scipy.stats import norm as normal

def get_gamma_pdf(x, alpha, beta):
  '''
    Return the pdf of custom gamma function on given points.
  '''
  return 1.0 / (beta**alpha * gamma(alpha)) * x**(alpha-1) * exp(-x/beta)

def gamma_rvs(alpha, beta, N=1):
  '''
    Return samples of the custom inverse Gamma RV.
  '''
  r = arange(0,300.1,0.1)
  gamma_pdf = get_gamma_pdf(r, alpha, beta)
  gamma_pdf = gamma_pdf / sum(gamma_pdf)
  if N == 1:
    return choice(r, p=gamma_pdf)
  else:
    return choice(r, size=N, p=gamma_pdf)

def get_beta_tau2_posterior_samples_MCMC(X, sigma2, n):
  '''
    Get the posterior samples for beta and tau2
  '''
  N = 20000
  # Posteriors before selection
```

```
  beta_j = zeros(N)
  tau2_j = zeros(N)
  # Initial guesses
  beta_j[0] = randn()*2 + 12
  tau2_j[0] = gamma_rvs(2.5, 2)
  # Generating loop
  j = 1
  while j < N:
    # Generate new sample
        beta = randn()*2 + 12
    tau2 = gamma_rvs(2.5, 2)
    # Determine whether to keep
    numerator = (randn()*sqrt(sigma2/n+tau2) + beta) *
     ↪  (randn()*sqrt(sigma2/n+tau2) + beta)
    denominator = (randn()*sqrt(sigma2/n+tau2_j[j-1]) + beta_j[j-1]) *
     ↪  (randn()*sqrt(sigma2/n+tau2_j[j-1]) + beta_j[j-1])
    r = min(numerator/denominator, 1)
    # Keep with probability r
    if rand() > r: continue
    beta_j[j] = beta
    tau2_j[j] = tau2
    j += 1
  # Discard the first 500 to minimize the influence of the initial guess
  beta_j = beta_j[500:]
  tau2_j = tau2_j[500:]
  return beta_j, tau2_j

def main():
  set_printoptions(precision=3)
  X = [array([12.7,6.6,14.7,12.2,4.4,7.8,13.8,13.7,11.1,9.1,14.0]),
    array([17.1,11.9,12.7,16.8,15.0,14.6,13.7,16.4]),
    array([5.2,4.5,10.5,15.0,5.0,14.9,7.6,8.3,10.8,14.6,15.1,7.0,9.3]),
    array([14.3,16.2,10.0,13.1,16.9,11.2,10.1,18.3,13.5,15.0,15.1,14.8,15.7,13.2,12.2,13
    array([10.5,7.5,4.7,12.5,13.1,13.5,12.2,16.1,9.0,17.9])]
  sigma2 = 4.
  n = 8.
  # Get the posteriors
  beta_post, tau2_post = get_beta_tau2_posterior_samples_MCMC(X, sigma2, n)
  # Calculate CDF
  x2_mean = mean(X[1])
  x3_mean = mean(X[2])
  m2 = (sigma2*beta_post/n + tau2_post*x2_mean) / (sigma2/n + tau2_post)
  m3 = (sigma2*beta_post/n + tau2_post*x3_mean) / (sigma2/n + tau2_post)
  v2 = (sigma2*tau2_post/n) / (sigma2/n + tau2_post)
  rv = normal()
```

```
  for b in [0, 1, 3, 5]:
    Ps = rv.cdf( (m2 - m3 - b) / sqrt(sigma2*2 + v2*2) )
    print "For b = %d, P = %.3f" % (b, mean(Ps))

if __name__ == '__main__':
  main()
```