

# Homework 2: Bayesian Inference

10/13/2025

5 Points Possible

Attempt 1



In Progress

**NEXT UP: Submit Assignment**

Add Comment

## Unlimited Attempts Allowed

10/6/2025 to 10/15/2025



### ▼ Details

#### Background

In lecture, we learned about Bayes' rule, which underlies so-called Bayesian machine learning, artificial intelligence, and statistics. To a "Bayesian", the correct way to analyze data is to postulate a stochastic model that you think produced the data (this is called the likelihood), then postulate a stochastic model for the parameters that you are trying to estimate (the prior). Then you observe the data (the evidence) and use Bayes' rule to obtain a posterior distribution on the model parameters. Note that this is a **distribution**: you don't get just one value for the estimated model parameters, but an entire distribution of them. Bayesians like this because it shows you the uncertainty as to the value of the model parameters.

However, one big problem with the Bayesian approach is that it can be really difficult to apply Bayes' rule to obtain the posterior distribution. Often the math can be nasty, especially if there is more than one variable that you are estimating. But there are two key tools that every Bayesian uses to make this easy. One is the *conjugate prior*. And the other is a huge toolbox of fancy sampling algorithms, like *Gibbs sampling*.

This homework has two tasks. In the first one, you'll be asked to use the idea of a conjugate prior to solve some inference problems, and in the second task you'll be asked to use a Gibbs sampler to perform Bayesian inference to estimate multiple parameters, for a real data set.

Before we begin, [here \(https://canvas.rice.edu/courses/80294/files/6680214?wrap=1\)](https://canvas.rice.edu/courses/80294/files/6680214?wrap=1)  [https://canvas.rice.edu/courses/80294/files/6680214/download?download\\_frd=1](https://canvas.rice.edu/courses/80294/files/6680214/download?download_frd=1) is the data that you'll use in this homework. This is a data set describing a population of 1000 young adults (18 years old). Each line of the file has a data point identifier, as well as height, in inches, and a weight, in pounds. Here is some [Python code \(https://canvas.rice.edu/courses/80294/files/6895433?wrap=1\)](https://canvas.rice.edu/courses/80294/files/6895433?wrap=1)  [https://canvas.rice.edu/courses/80294/files/6895433/download?download\\_frd=1](https://canvas.rice.edu/courses/80294/files/6895433/download?download_frd=1) that you can use to start with. This is a stub that you'll build on for this assignment. Note that this stub will read in this file, and put (height, weight) pairs into a dictionary, where the key is the identifier (from 1 to 1000) and the value is a (height, weight) tuple.

#### Task 1

Now, imagine that we have a stochastic model that says that the way that a person's weight is generated is:

$$weight \sim \text{Normal}(\mu, \sigma)$$

where:

$$\mu = height \times m + c$$

Note that there are three parameters to this model:  $m, c, \sigma$ . We'd like to be able to estimate the values of these parameters for this data set, using the Bayesian approach. In this part of the assignment, you'll write Python functions that use Bayes' rule to sample from the posterior distributions of each of these parameters separate.

### Task 1.1

(1.25 pts.) Let's assume that  $m$  is 20,  $c$  is 50, but we don't know  $\sigma^2$ . Assume that there is an InverseGamma (10.0, 1.0) prior on  $\sigma^2$  (the [inverse gamma distribution](https://en.wikipedia.org/wiki/Inverse-gamma_distribution)  $\Rightarrow$  [https://en.wikipedia.org/wiki/Inverse-gamma\\_distribution](https://en.wikipedia.org/wiki/Inverse-gamma_distribution)) takes two parameters: a shape and a scale, often given as  $\alpha$  and  $\beta$ ). That is,  $P(\sigma^2)$  is InverseGamma (10.0, 1.0).

Note that given the way a person's weight is generated can be re-written as  $weightDiff \sim \text{Normal}(0, \sigma)$  where  $weightDiff = weight - (height \times m + c)$

Here,  $weightDiff$  is the difference between the observed weight and the expected weight. This is mathematically the same as before, but now we have  $weightDiff \sim \text{Normal}(0, \sigma)$  as the likelihood function. That is,  $P(weightDiff|\sigma)$  is  $\text{Normal}(0, \sigma)$ .

Take some time to understand the above math.

### Coding Task

Now here's where it's time to code. Write a Python function called `sample_sigma` that is able to take a single sample from the posterior distribution for  $\sigma$ , given this prior and this likelihood. In your results file, include the results of calling your `sample_sigma` function 10 times. Here's what I got:

```
1200.6871167281322, 1251.5588536723492, 1303.6372247966158, 1223.3700793019784,
1257.4637882502, 1252.3520805623205, 1283.489892089508, 1228.9545878729455,
1263.0357828310423, 1267.5842006438613
```

You may get slightly different results, given that this is random.

### Some hints:

(1) The Inverse Gamma distribution is a conjugate prior for the variance of a Normal distribution with known mean (zero in this case), so it's quite easy to compute the posterior distribution here. Once you figure out what a "conjugate prior" is, it's just looking up how to use the conjugacy on [Wikipedia](https://en.wikipedia.org/wiki/Conjugate_prior)  $\Rightarrow$  [https://en.wikipedia.org/wiki/Conjugate\\_prior](https://en.wikipedia.org/wiki/Conjugate_prior), and then implementing the formula that you find there.

(2) To sample from an Inverse Gamma distribution in Python, use

```
"temp = invgamma.rvs (a = shapeYouWant, scale = scaleYouWant)".
```

Make sure to import the Inverse Gamma distribution: `from scipy.stats import invgamma`.

## Task 1.2

(1.25 pts.) Let's assume that  $m$  is 20,  $\sigma$  is 200. Assume that there is a Normal (50, 100) prior on  $c$ . Write a Python function called `sample_c` that is able to take a single sample from the posterior distribution for  $c$ . In your results file, include the results of calling your `sample_c` function 10 times.

Note that since  $weight \sim \text{Normal}(\mu, \sigma)$  where  $\mu = height \times m + c$ , we can re-write this  $weightDiff \sim \text{Normal}(c, \sigma)$  where this time  $weightDiff = weight - height \times m$ .

Thus, the likelihood  $P(weightDiff|c)$  is  $\text{Normal}(c, \sigma)$  and you can use the fact that the Normal is the conjugate prior for the mean of the Normal with known variance to implement this.

Note that to sample from a Normal distribution, you will want to use the Python "`import numpy as np`" and then to sample from a Normal with a given mean and standard deviation, use:

`"np.random.normal (mean, stdDev) "`.

## Task 1.3

(1.25 pts.) Let's assume that that  $c$  is 50,  $\sigma$  is 200.

Assume that there is a Normal (5, 10) prior on  $m$ . Write a Python function called `sample_m` that is able to take a single sample from the posterior distribution for  $m$ .

Note that since  $weight \sim \text{Normal}(\mu, \sigma)$  where  $\mu = height \times m + c$ , we can re-write this  $weightDiff \sim \text{Normal}(m, \sigma/height)$  where this time  $weightDiff = \frac{(weight-c)}{height}$ . Thus, the likelihood  $P(weightDiff|m)$  is  $\text{Normal}(m, \sigma/height)$ .

This one is a little different from the last one, as the height is different for every person, so that the likelihood function has a different standard deviation/variance for every person. Thus you can't directly use the Wikipedia formula that applies when you have a Normal prior on the mean of a Normal with known variance. The Wikipedia formula assumes that the variance/standard deviation is the same for each observed data point. Fortunately, a slight modification to the Wikipedia formula works in this case.

Let's assume that we generalize the Wikipedia formula so that we have a Normal likelihood where the  $i$ th data point is  $x_i$ , but now we allow the standard deviation of the Normal distribution that produced the  $i$ th data point to be  $\sigma/\alpha_i$  for some known value of  $\alpha_i$ . The posterior mean and variance in this generalized case become:

$$\frac{1}{\frac{1}{\sigma_0^2} + \sum_{i=1}^n \frac{\alpha_i^2}{\sigma^2}} \left( \frac{\mu_0}{\sigma_0^2} + \frac{\sum_{i=1}^n x_i \alpha_i^2}{\sigma^2} \right), \left( \frac{1}{\sigma_0^2} + \sum_{i=1}^n \frac{\alpha_i^2}{\sigma^2} \right)^{-1}$$

It's easily possible to use this new rule to implement `sample_m`. Same as before, include the results of calling your `sample_m` function 10 times in your results file.

## Task 2

(1.25 pts.) Once you have completed Task 1, Task 2 is really easy. In Bayesian statistics/machine learning, We often run into cases where there is not just one variable we are trying to estimate, but several. In our example, we have three:  $m$ ,  $c$ ,  $\sigma$ . In Part 1, we sampled each of them separately, but now let's say we want to estimate them all together, each with their own prior, as described above. How to do this?

There are a number of classic algorithms that apply; one is called **Gibbs sampling**. In Gibbs sampling, you first guess values for all of the variables you are trying to estimate. Let's start with  $m = 20$ ,  $c = 50$ ,  $\sigma = 200$  as in the Python stub. Then, you just loop for a long time (say 1k iterations), and cycle through all of the variables, sampling them one-at-a-time. When you sample each variable, you assume that the others are fixed and so you only have to consider single-variable posteriors.

Our our case, the Gibbs sampling algorithm is:

```
for x in range(1000):
    get_error ();
    sigma = sample_sigma ();
    m = sample_m ();
    c = sample_c ();
```

There is classic statistical theory that says that when you run this for a long time, the current value for the triple  $m$ ,  $c$ ,  $\sigma$  will actually be a sample from the **joint posterior distribution** for all three. Thus, Gibbs sampling gives you a way to sample from complex, joint posteriors.

Note that the call to `get_error` is not part of the Gibbs sampler, but it will allow you to watch the progress of the sampler. As the sampler produces better and better samples from the joint posterior, the error will go down.

To get credit for Task 2, simply include in your results file submission (a) the first five errors, (b) the last five errors, and (c) the final  $m$ ,  $c$ ,  $\sigma$  values that you get from your Gibbs sampler.

## Submission

Submit a single **netid\_homework2.zip** compressed file containing your code in a **netid\_hw2code.py** file and your outputs in a **netid\_hw2results.txt** file.

### Choose a submission type

