**STATS370, Fall 2022**
**Bayesian Statistics**
**Julian Cooper**

**Project: Sampling Methodologies**

**Purpose** of the project is to compare different sampling methodologies for estimating the posterior $p(\theta|y_1, ..., y_n)$ of a bayesian inference problem.

**Data** $(y_1, ..., y_n)$ are gene expression measurements for two genes on $n = 24$ samples where $y_i = (y_{i,1}, y_{i,2})$ represent gene expressions for sample i. Our samples are all labelled by "group", denoted $(t_1, ..., t_n)$. Cell type (or mix of cell types) vary with group and we assume mean gene expression (but not variance) depends on cell type. Moreover, the expressions of different genes are independently normally distributed.

- $Y_i \sim N(\mu, \sigma^2\mathbb{I})$ if sample $i$ from group 1
- $Y_i \sim N(\gamma, \sigma^2\mathbb{I})$ if sample $i$ from group 2
- $Y_i \sim N(0.5\mu + 0.5\gamma, \sigma^2\mathbb{I})$ if sample $i$ from group 3
- $Y_i \sim N(\tau\mu + (1-\tau)\gamma, \sigma^2\mathbb{I})$ if sample $i$ from group 4

Our model has a 6-dimensional parameter $\theta = (\sigma^2, \tau, \mu_1, \mu_2, \gamma_1, \gamma_2)$.

For all sampling methodologies, I computed 5000 samples with a burn-in of 200. The rest (step size, momentum, proposals, etc.) I discuss and treat as design choices.

# 1 Metropolis Hastings

To implement the Metropolis-Hastings algorithm we require the joint and transition densities to compute the hastings ratio. Below we derive the join density for our model, and treat the transition density as a design choice.

**Joint density**:

$$\begin{aligned} p(\theta|y) &\propto \prod_{i=1}^{n} p(y|\theta)p(\theta) \\ &= \prod_{t=1}^{4}\prod_{i=1}^{n} 1\{t_i = t\} \cdot p(y_i|\theta) \cdot p(\mu) \cdot p(\gamma) \cdot p(\tau) \cdot p(\sigma^2) \\ &= \prod_{t_i=1} N(\mu, \sigma^2\mathbb{I}) \cdot \prod_{t_i=2} N(\gamma, \sigma^2\mathbb{I}) \cdot \prod_{t_i=3} N(0.5\mu + 0.5\gamma, \sigma^2\mathbb{I}) \cdot \prod_{t_i=4} N(\tau\mu + (1-\tau)\gamma, \sigma^2\mathbb{I}) \cdot p(\sigma^2) \end{aligned}$$

**Hastings Ratio**: let $q(\theta_0, \theta_1)$ be our transition density and $\pi(y, \theta)$ be our join density. Then,

$$\text{accept prob.} = \min(1, \frac{q(\theta_0, \theta_1)\pi(y, \theta_1)}{q(\theta_1, \theta_0)\pi(y, \theta_0)})$$

```
def metropolis_hastings(data, n_samples, step_size, inital_position):
    curr_theta = inital_position.copy()
    samples = [curr_theta]

    while it < n_samples:

        proposed = proposal_sampler(curr_theta, step_size)
        h_ratio = hastings_ratio(proposed, curr_theta, data)
        accept_prob = min(1, h_ratio)

        if np.random.uniform(0, 1) <= accept_prob:
            curr_theta = proposed
            accept_count += 1

        samples.append(curr_theta)

    return samples
```

The Metropolis-Hastings algorithm itself is very simple, However, there are still a number of design choices we need to make, such as (a) how to propose a new sample, (b) enforcing any constraints we have (e.g. $\tau \in [0, 1]$), and (c) choosing hyperparameters (step size, burn-in).

(a) **Proposal selection**: Tried two methods: symmetric with constraints imposed as part of acceptance criteria, and asymmetric with constraints encoded into proposal distributions.

The symmetric method was attractive from an implementation perspective, since it would guarantee detail balance (symmetric transition kernel) and therefore simplify our hastings ratio to $\frac{\pi(y,\theta_1)}{\pi(y,\theta_0)}$ since $q(\theta_0, \theta_1) = q(\theta_1, \theta_0)$. I picked a gaussian random step with mean equal to step size and variance of 1.

```
def proposal_sampler(curr_theta, step_size):
    num_params = len(curr_theta)

    while True:
        randomness = np.random.normal(0, 1, num_params)
        proposal = curr_theta + np.multiply(step_size, randomness)

        if proposal[0] > 0 and (proposal[1] >= 0 and proposal[1] <=
1):
            break

    return proposal
```

The asymmetric method made use of truncated normals that directly encoded the bounds of our problem in the proposal distributions. This required that I also solve for an asymmetric transition kernel and evaluate the full hastings ratio, but the resulting algorithm was much more efficient (see constraints section below).

```
def proposal_sampler(curr_theta, step_size):
    proposed = np.zeros_like(curr_theta)
    a, b = (0 - proposed[0])/step_size, (np.inf - proposed[0])/step_size
    c, d = (0 - proposed[1])/step_size, (1 - proposed[1])/step_size

    proposed[0] = truncnorm(a, b, proposed[0], step_size).rvs()
    proposed[1] = truncnorm(c, d, proposed[1], step_size).rvs()
    proposed[2] = norm(proposed[2], step_size).rvs()
    proposed[3] = norm(proposed[3], step_size).rvs()
    proposed[4] = norm(proposed[4], step_size).rvs()
    proposed[5] = norm(proposed[5], step_size).rvs()

    return proposal
```

(b) **Enforcing constraints**: We had two constraints to consider: (a) $\sigma^2 > 0$ and (b) $0 \leq \tau \leq 1$. I enforce these constraints for our symmetric proposal method by resampling if the proposal violates either (a) or (b). This reduces the efficiency (acceptance ratio) of the algorithm substantially and introduces potential bias in our resulting marginal posterior samples (particularly for $\tau$ which has significant density near its upper bound, mean shifted from 0.75 to 0.85). By contrast, our asymmetric proposal method encodes these constraints directly. For these reasons, we use the asymmetric kernel for our results.

(c) **Hyperparameters**: There are not many hyperparameters to tune for Metropolis-Hastings, which is one of its advantages! I experimented with different step sizes (0.01, 0.05, 0.1, 0.5), and for each computed the acceptance ratio and number of effective (uncorrelated) samples for 300 sample test (after burn-in). I choose a step size of 0.05 since it maximized both number of effective samples and acceptance ratio.

| Step size | 0.01 | 0.05 | 0.1 | 0.5 |
|---|---|---|---|---|
| Acceptance ratio | 0.87 | 0.44 | 0.22 | 0.02 |
| Mean effective samples | 3.5 | 16.6 | 14.6 | 6.96 |

I did not investigate varying initialization (used $[1., 0.5, 0., 0., 0., 0.]$) or burn-in (fixed at 200 to match what I saw in textbook examples).

**Results**: Using the above design choices, we produce the following samples from our posterior (figure 1) and compute the first and second moments (table below). These passed basic sense checks, such as $\sigma^2$ and $\tau$ falling within their respective bounds, mean and variance matching results from other sampling algorithms, and lower variance as we increase number of samples.

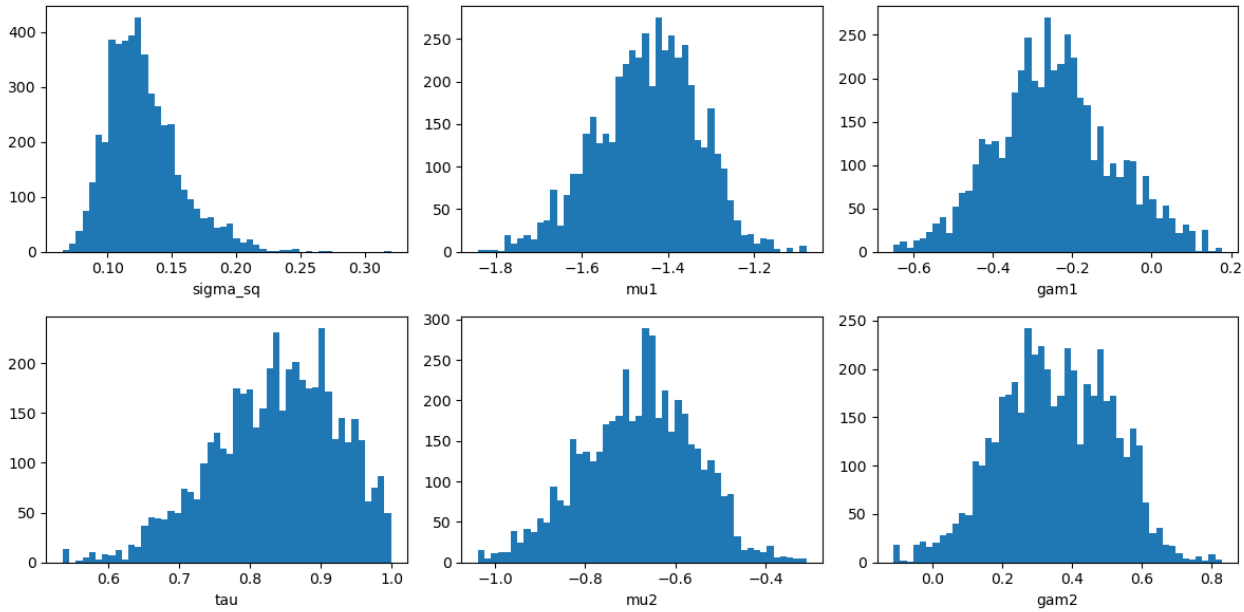


Figure 1: Histogram of Metropolis-Hastings posterior samples

| | sigma | tau | mu1 | mu2 | gam1 | gam2 |
|---|---|---|---|---|---|---|
| Mean of posterior samples | 0.14 | 0.84 | -1.44 | -0.67 | -0.25 | 0.35 |
| Variance of posterior samples | 0.004 | 0.008 | 0.015 | 0.023 | 0.022 | 0.025 |

The traceplots in figure 2 give an indication of how well the sampler explored the posterior density. I am comforted by the fact that there are no long periods of repeated values (getting stuck) despite an acceptance rate of only 0.44.

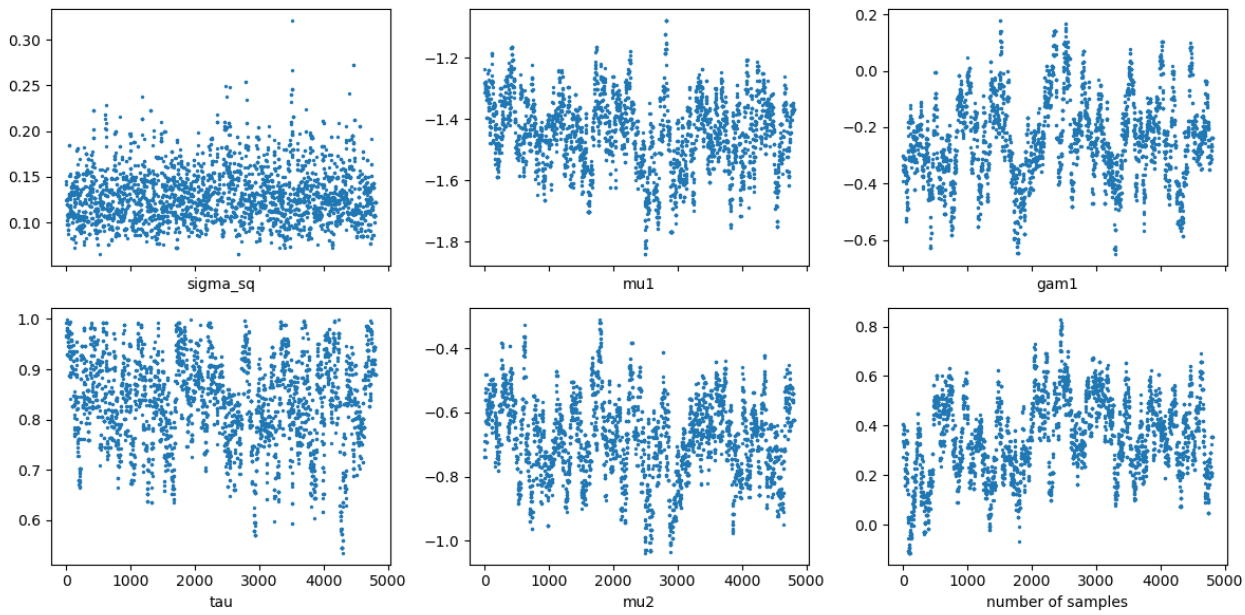


Figure 2: Traceplot for Metropolis-Hastings sampling

**Final remarks**. Metropolis-Hastings is simple to implement and relatively fast to run (time = 197 sec for 5000 samples), however, number of effective samples (autocorrelation) and acceptance ratio were not as good as we might expect from other technqiues that take more directed paths through the posterior space (e.g. HMC).

# 2 Hamiltonian Monte Carlo

To implement the Hamiltonian Monte Carlo we need to compute the Hamiltonian $H(\rho, \theta)$ given independently drawn momentum $\rho$ and our current paramewter values $\theta$.

$$\begin{aligned} H(\theta, y) &= -\log p(\rho, \theta) \\ &= -\log p(\rho|\theta) - \log p(\theta) \\ &= K(\rho, \theta) + V(\theta) \qquad \text{kinetic and potential energy} \end{aligned}$$

We now evolve the system using the following equations from Hamiltonian mechanics. Note: $\frac{\partial V}{\partial \rho} = 0$ and if we choose our kinetic energy to be gaussian $K(\rho, \theta) = \frac{1}{2}\rho^T M^{-1}\rho + \log|M| +$ const. so we get $\frac{\partial K}{\partial \rho} = M^{-1}\rho$ and $\frac{\partial K}{\partial \theta} = 0$.

$$\begin{aligned} \frac{d\theta}{dt} &= +\frac{\partial H}{\partial \rho} = +\frac{\partial K}{\partial \rho} + \frac{\partial V}{\partial \rho} = M^{-1}\rho \\ \frac{d\rho}{dt} &= -\frac{\partial H}{\partial \theta} = -\frac{\partial K}{\partial \theta} - \frac{\partial V}{\partial \theta} = -\frac{\partial V}{\partial \theta} \end{aligned}$$

Therefore, to implement the above system, we need only to calculate the gradient of our potential energy (negative log likelihood) with respect to our parameters $\theta$.

**Negative log likelihood**: let $V(\theta) = -l(\theta) = -\log p(\theta|y)$,

$$\begin{aligned}
V(\theta) &= -\log \prod_{i=1}^{n} p(y|\theta)p(\theta) \\
&= -\sum_{i=1}^{n} \log p(y|\theta) - \log p(\theta) && \text{let } y_i' = \text{centered data} \\
&= -\sum_{i=1}^{n} \log \left[ (2\pi)^{-k/2} \cdot |\sigma^2 \mathbb{I}|^{-1/2} \cdot \exp(\frac{-1}{2} y_i'^T (\sigma^2 \mathbb{I})^{-1} y_i') \right] - \log \frac{1}{\sigma^2} \\
&= \frac{nk}{2} \log 2\pi + (n+1) \log \sigma^2 + \frac{1}{2\sigma^2} \sum_{i=1}^{n} y_i'^T \mathbb{I}^{-1} y_i'
\end{aligned}$$

**Gradient of $V(\theta)$ wrt $\theta$**:

$$\begin{aligned}
\frac{\partial V(\theta)}{\partial \sigma^2} &= \frac{n+1}{\sigma^2} - \frac{1}{2(\sigma^2)^2} \sum_{i=1}^{n} \|y_i'\|_2^2 \\
\frac{\partial V(\theta)}{\partial \tau} &= \frac{1}{\sigma^2} \cdot (\gamma - \mu)^T \sum_{t_i=4} y_i' \\
\frac{\partial V(\theta)}{\partial \mu} &= -\frac{1}{\sigma^2} \cdot \left[ \sum_{t_i=1} y_i' + 0.5 \sum_{t_i=3} y_i' + \tau \sum_{t_i=4} y_i' \right] \\
\frac{\partial V(\theta)}{\partial \gamma} &= -\frac{1}{\sigma^2} \cdot \left[ \sum_{t_i=2} y_i' + 0.5 \sum_{t_i=3} y_i' + (1-\tau) \sum_{t_i=4} y_i' \right]
\end{aligned}$$

Our implementation of these hamiltonian equations employs a leapfrog algorithm to perform numerical integration. This is a second-order method as opposed to euler which is first-order. This design choice helps with stability.

```
def leapfrog(q, p, data, M_mat, path_len, step_size):
    q, p = np.copy(q), np.copy(p)  # curr_theta = q

    p -= step_size * dVdq(data, q) / 2
    for _ in range(int(path_len / step_size)):
        q, p = q_update(q, p, M_mat, step_size)
        p -= step_size * dVdq(data, q)

    q, p = q_update(q, p, M_mat, step_size)
    p -= step_size * dVdq(data, q) / 2

    # momentum flip at end
    return q, -p
```

```python
def hamiltonian_monte_carlo(n_samples, initial_position, m, step_size, path_len):
    samples, theta = [], inital_position
    size = (n_samples,) + initial_position.shape[:1]
    M_mat = np.eye(len(initial_position)) * m
    momentum = st.norm(0, m)

    for p0 in momentum.rvs(size=size):
        # integrate over our path to get new position and momentum
        q_new, p_new = leapfrog(
            theta,
            p0,
            data,
            M_mat=M_mat,
            path_len=path_len,
            step_size=step_size,
        )

        # check metropolis acceptance criterion
        curr_u = -log_likelihood(data, theta) #- beta(20, 3).logpdf(theta[1])
        prop_u = -log_likelihood(data, q_new) #- beta(20, 3).logpdf(q_new[1])
        curr_k = (0.5/m)*np.linalg.norm(p0,2)**2
        prop_k = (0.5/m)*np.linalg.norm(p_new,2)**2
        log_hratio = curr_u - prop_u + curr_k - prop_k
        accept_log_prob = min(0, log_hratio)

        if np.log(np.random.uniform(0, 1)) <= accept_log_prob:
            samples.append(q_new)
            accept_count += 1
            theta = q_new
        else:
            samples.append(np.copy(samples[-1]))

    return np.array(samples)
```

The Hamiltonian Monte Carlo algorithm has a number of design choices, including (a) whether to use an acceptance criterion, (b) enforcing any constraints we have (e.g. $\tau \in [0, 1]$), and (c) choosing hyperparameters (step size, momentum).

(a) **Acceptance criterion**: My first implementation did not include an acceptance step which was faster to evaluate but did not produce sensible, unbiased posterior samples. Adding the metropolis acceptance criterion at the end of each iteration helped to correct for errors introduced via numerical integration.

(b) **Enforcing constraints**: We had two constraints to consider: (a) $\sigma^2 > 0$ and (b) $0 \leq \tau \leq 1$. I experimented with two methods of enforcing these constraints: boundary reflection (from particle physics applications) and imposing a beta prior on $\tau$.

For the boundary reflection method, I enforce these constraints in the q ($\theta$) update function by creating a "hard wall" or barrier, where we assume both $p$ (momentum) and $q$ (parameter $\theta$) are reflected (flip sign) by same magnitude that they surpassed

the constraint. While this method did seem to work (resulting posterior marginals had similar first and second moments), it was less efficient (lower effective acceptance ratio).

```python
def q_update(q, p, M_mat, step_size):
    """Helper function to update theta within bounds"""
    q_prop = q + step_size * np.linalg.inv(M_mat) @ p

    # check bounds
    if q_prop[0] < 0:
        p[0] = -p[0]
        q_prop[0] = -q_prop[0]

    if q_prop[1] < 0 or q_prop[1] > 1:
        p[1] = -p[1]
        q_prop[1] = -q_prop[1]

    return q_prop, p
```

Imposing a beta prior on $\tau$ (Beta($\alpha = 20, \beta = 3$)) not only encoded the constraints we wanted into our problem but also made the resulting joint posterior space differentiable, and therefore easier to traverse and sample from with gradient steps. This was the method I ended up using to produce my results. Note, this requied we both update our log likelihood and gradient calculations to reflect the new beta prior (rather than Unif[0,1]).

$$V(\theta) = \frac{nk}{2}\log 2\pi + (n+1)\log\sigma^2 + \frac{1}{2\sigma^2}\sum_{i=1}^{n} y_i'^T \mathbb{I}^{-1} y_i' + \log(\mathrm{B}(\alpha,\beta)) - \log(\tau^{\alpha-1}(1-\tau)^{\beta-1})$$

$$\frac{\partial V(\theta)}{\partial \tau} = \frac{1}{\sigma^2}\cdot(\gamma-\mu)^T \sum_{t_i=4} y_i' + \frac{1-\alpha}{\tau} + \frac{\beta-1}{1-\tau}$$

(c) **Hyperparameters**: I experimented with different step sizes (0.005, 0.01) and momentum variance values (1, 5, 10). For each combination I computed the acceptance ratio and number of effective (uncorrelated) samples for 500 sample test (after burn-in of 200). I choose a (step size, momentum) pair of (0.01, 5) since it maximized number of effective samples while maintaining reasonable acceptance. Note, for higher step sizes (e.g 0.05) my algorithm would run into numerical instability issues so I did not test further in this direction.

| Step size | Momentum | Accept ratio | Mean eff. samples |
|---|---|---|---|
| 0.005 | 1 | 0.97 | 138 |
| 0.005 | 5 | 0.83 | 159 |
| 0.005 | 10 | 0.80 | 30 |
| 0.01 | 1 | 0.92 | 124 |
| 0.01 | 5 | 0.76 | 195 |
| 0.01 | 10 | 0.69 | 10 |

I did not investigate varying initialization (used $[1., 0.8, 0., 0., 0., 0.]$), burn-in (fixed at 200 to match what I saw in textbook examples) or path length (used 1 as set number of leapfrog steps eual to path length / step size).

**Results**: Using the above design choices, we produce the following samples from our posterior (figure 3).

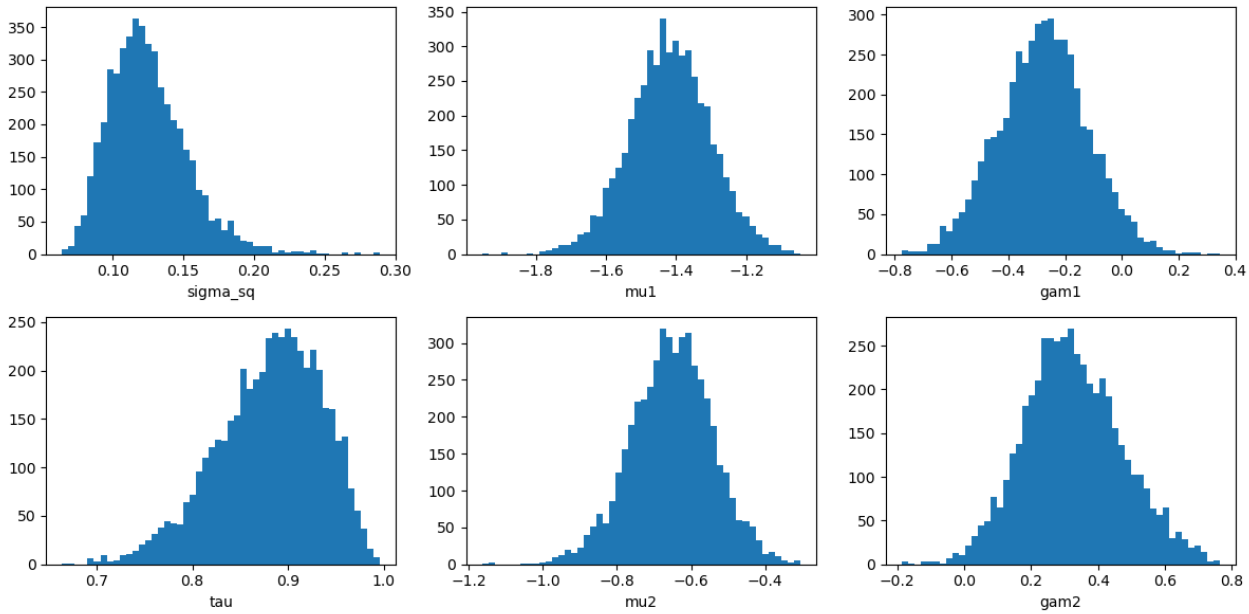


Figure 3: Histogram of Hamiltonian Monte Carlo posterior samples

| | sigma | tau | mu1 | mu2 | gam1 | gam2 |
|---|---|---|---|---|---|---|
| Mean of posterior samples | 0.13 | 0.88 | -1.42 | -0.65 | -0.28 | 0.33 |
| Variance of posterior samples | 0.001 | 0.003 | 0.013 | 0.012 | 0.024 | 0.022 |

The traceplots in figure 4 give an indication of how well the sampler explored the posterior density. I am comforted by the fact that there are no long periods of repeated values (getting stuck) despite us not implementing no-turn logic.

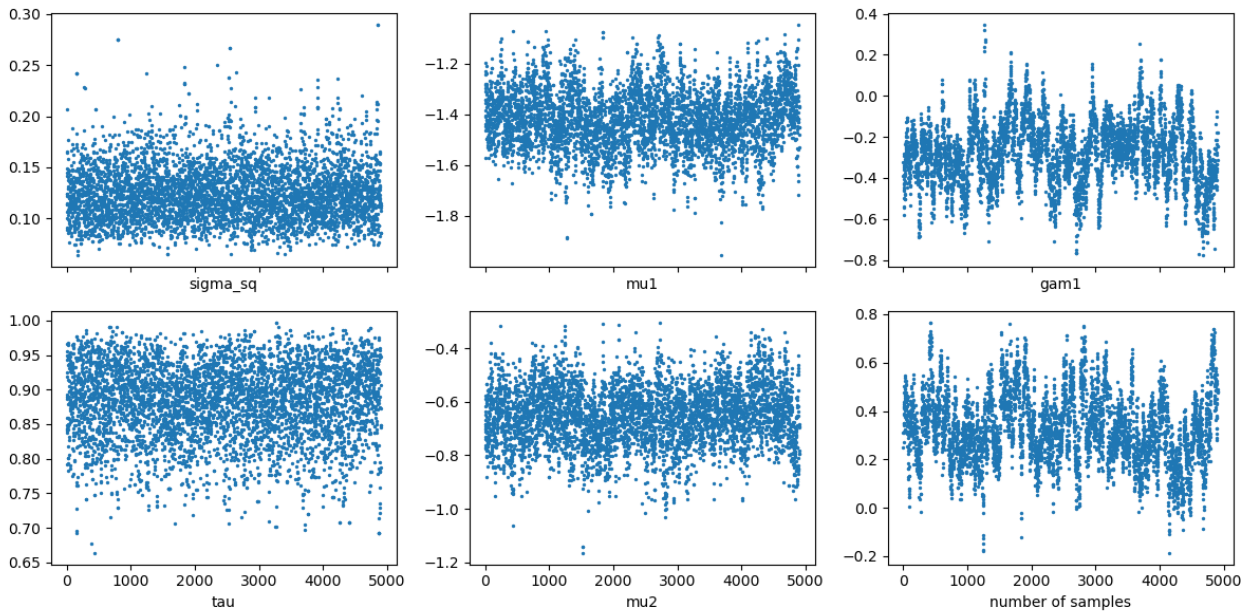


Figure 4: Traceplot for Hamiltonian Monte Carlo sampling

**Final remarks**. HMC was much slower to run than Metropolis- Hastings due to the gradient evaluation at each iteration (4,205 sec for 5000 samples), however, our acceptance ratio was much higher (approx. 0.85 vs 0.45). Moreover, our samples were far less likely to be autocorrelated (approx. 32 vs 5 effective samples per 100 iterations). This is because we are taking directed rather than random steps through the posterior.

# 3 Gibbs Sampling

To implement the Gibbs Sampling we need to derive the conditional distributions for each parameter from which to sample.

**Conditional distributions** Idea: start with joint posterior density, then remove constants (e.g. terms with only fixed parameters) and rearrange to derive density conditional distribution we can sample from. To simplify notation, let $\theta[-x]$ represent all parameters of $\theta$ exluding parameter $x$ and let $y_i'$ represent centered data (e.g. $y_1' = y_1 - \mu$).

Derive conditional distribution for $\sigma^2$.

$$p(\sigma^2|y, \theta[-\sigma^2]) \propto \prod_{t_i=1} N(\mu, \sigma^2\mathbb{I}) \cdot \prod_{t_i=2} N(\gamma, \sigma^2\mathbb{I}) \cdot \prod_{t_i=3} N(0.5\mu + 0.5\gamma, \sigma^2\mathbb{I}) \cdot \prod_{t_i=4} N(\tau\mu + (1-\tau)\gamma, \sigma^2\mathbb{I}) \cdot p(\sigma^2)$$

$$\propto |\sigma^2\mathbb{I}_2|^{-n/2} \cdot \frac{1}{\sigma^2} \cdot \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n} y_i'^T \mathbb{I}^{-1} y_i'\right)$$

$$
\begin{aligned}
&\propto (\frac{1}{\sigma^2})^{n+1} \cdot \exp\left(-\frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^{n} \|y_i'\|_2^2\right) \\
&= x^{\alpha+1} \cdot \exp(-x\beta) \\
\therefore \sigma^2|y, \theta[-\sigma^2] &\sim \text{InvGamma}\left(n, \quad \frac{1}{2} \sum_{i=1}^{n} \|y_i'\|_2^2\right)
\end{aligned}
$$

Derive conditional distribution for $\tau$.

$$
\begin{aligned}
p(\tau|y, \theta[-\tau]) &\propto \exp\left(-\frac{1}{2\sigma^2} \sum_{t_i=4} (y_i - (\tau\mu + (1-\tau)\gamma))^T \mathbb{I}^{-1} (y_i - (\tau\mu + (1-\tau)\gamma))\right) \\
&\propto \exp\left(-\frac{1}{2} \cdot \frac{\|\gamma - \mu\|_2^2}{\sigma^2} \sum_{t_i=4} (\tau\vec{i} - \alpha_i)^T \mathbb{I}^{-1} (\tau\vec{i} - \alpha_i)\right) && \text{let } \vec{i} = (1,1) \\
&\propto \exp\left(-\frac{1}{2} \cdot \frac{\|\gamma - \mu\|_2^2}{\sigma^2} \sum_{t_i=4} \left(\|\tau\vec{i}\|_2^2 - 2\tau\vec{i}^T\alpha_i + \|\alpha_i\|_2^2\right)\right) && \|\alpha_i\|_2^2 \text{ constant} \\
&\propto \exp\left(-\frac{1}{2} \cdot \frac{\|\gamma - \mu\|_2^2}{\sigma^2} \left(n_4\tau^2 - 2\tau\vec{i}^T \sum_{t_i=4} \alpha_i\right)\right) \\
&\propto \exp\left(-\frac{1}{2} \cdot \frac{n_4\|\gamma - \mu\|_2^2}{\sigma^2} \left(\tau - \frac{\sum_{t_i=4} \alpha_i}{n_4}\right)^2\right) && \text{complete the square} \\
\therefore \tau|y, \theta[-\tau] &\sim \text{TruncNormal}\left(\frac{\sum_{t_i=4} \alpha_i}{n_4}, \frac{\sigma^2}{n_4\|\gamma - \mu\|_2^2}, 0, 1\right) \quad \text{where } \alpha_i = (y_i - \gamma)^T(\mu - \gamma)
\end{aligned}
$$

Derive conditional distribution for $\mu$.

$$
\begin{aligned}
p(\mu|y, \theta[-\mu]) &\propto \exp\left(-\frac{1}{2\sigma^2} \sum_{t_i \in \{1,3,4\}} (y_i - \overline{y}_i)^T \mathbb{I}^{-1} (y_i - \overline{y}_i)\right) && \overline{y}_i = \text{group mean} \\
&\propto \exp\left(-\frac{1}{2} \cdot \frac{1}{\sigma^2} \sum_{t_i \in \{1,3,4\}} (\mu - \overline{\mu}_i^{t_i})^T \mathbb{I}^{-1} (\mu - \overline{\mu}_i^{t_i})\right) && \text{rearrange for } \mu \\
&\propto \exp\left(-\frac{1}{2} \cdot \frac{1}{\sigma^2} \sum_{t_i \in \{1,3,4\}} \left(\|\mu\|_2^2 - 2\mu^T\overline{\mu}_i^{t_i} + \|\overline{\mu}_i^{t_i}\|_2^2\right)\right) && \|\overline{\mu}_i^{t_i}\|_2^2 \text{ constant} \\
&\propto \exp\left(-\frac{1}{2} \cdot \frac{1}{\sigma^2} \left((n_1 + 0.5^2 n_3 + \tau^2 n_4)\|\mu\|_2^2 - 2\mu^T \sum_{t_i \in \{1,3,4\}} \overline{\mu}_i^{t_i}\right)\right) && \text{take } \|\mu\|_2^2 \text{ outside sum} \\
&\propto \exp\Bigg(-\frac{1}{2} \cdot \frac{n_1 + 0.5^2 n_3 + \tau^2 n_4}{\sigma^2} && \text{complete the square} \\
&\quad \left(\mu - \frac{\sum_{t_i \in \{1,3,4\}} \overline{\mu}_i^{\{t_i\}}}{n_1 + 0.5^2 n_3 + \tau^2 n_4}\right)^T \mathbb{I}^{-1} \left(\mu - \frac{\sum_{t_i \in \{1,3,4\}} \overline{\mu}_i^{\{t_i\}}}{n_1 + 0.5^2 n_3 + tau^2 n_4}\right)\Bigg)
\end{aligned}
$$

$$\therefore \mu|y,\theta[-\mu] \sim \text{Normal}\left(\frac{\sum_{t_i\in\{1,3,4\}}\overline{\mu}_i^{\{t_i\}}}{n_1+0.5^2n_3+\tau^2n_4}, \quad \frac{\sigma^2}{n_1+0.5^2n_3+\tau^2n_4}\right)$$
$$\text{where} \quad \overline{\mu}_i^{t=1}=y_i; \quad \overline{\mu}_i^{t=3}=0.5(y_i-0.5\gamma); \quad \overline{\mu}_i^{t=4}=\tau(y_i-(1-\tau)\gamma)$$

Derive conditional distribution for $\gamma$.

$$\begin{aligned}
p(\gamma|y,\theta[-\gamma]) &\propto \exp\left(-\frac{1}{2\sigma^2}\sum_{t_i\in\{2,3,4\}}(y_i-\overline{y}_i)^T\mathbb{I}^{-1}(y_i-\overline{y}_i)\right)\\
&\propto \exp\left(-\frac{1}{2}\cdot\frac{1}{\sigma^2}\sum_{t_i\in\{2,3,4\}}(\gamma-\overline{\gamma}_i^{t_i})^T\mathbb{I}^{-1}(\gamma-\overline{\gamma}_i^{t_i})\right)\\
&\propto \exp\left(-\frac{1}{2}\cdot\frac{1}{\sigma^2}\sum_{t_i\in\{2,3,4\}}\left(\|\gamma\|_2^2-2\gamma^T\overline{\gamma}_i^{t_i}+\|\overline{\gamma}_i^{t_i}\|_2^2\right)\right)\\
&\propto \exp\left(-\frac{1}{2}\cdot\frac{1}{\sigma^2}\left((n_2+0.5^2n_3+(1-\tau)^2n_4)\|\gamma\|_2^2-2\gamma^T\sum_{t_i\in\{2,3,4\}}\overline{\gamma}_i^{t_i}\right)\right)\\
&\propto \exp\Bigg(-\frac{1}{2}\cdot\frac{n_2+0.5^2n_3+(1-\tau)^2n_4}{\sigma^2}\\
&\quad\left(\gamma-\frac{\sum_{t_i\in\{2,3,4\}}\overline{\gamma}_i^{\{t_i\}}}{n_2+0.5^2n_3+(1-\tau)^2n_4}\right)^T\mathbb{I}^{-1}\left(\gamma-\frac{\sum_{t_i\in\{2,3,4\}}\overline{\gamma}_i^{\{t_i\}}}{n_2+0.5^2n_3+(1-\tau)^2n_4}\right)\Bigg)
\end{aligned}$$

$$\therefore \gamma|y,\theta[-\gamma] \sim \text{Normal}\left(\frac{\sum_{t_i\in\{2,3,4\}}\overline{\gamma}_i^{\{t_i\}}}{n_2+0.5^2n_3+(1-\tau)^2n_4}, \quad \frac{\sigma^2}{n_2+0.5^2n_3+(1-\tau)^2n_4}\right)$$
$$\text{where} \quad \overline{\gamma}_i^{t=2}=y_i; \quad \overline{\gamma}_i^{t=3}=0.5(y_i-0.5\mu); \quad \overline{\gamma}_i^{t=4}=(1-\tau)(y_i-\tau\mu)$$

Having derived our conditional distributions, the gibbs implementation itself is very straightforward. We iterate through and update one or multiple parameters at a time (see below).

```
def gibbs_sampling(data, n_samples, initial_position):
    samples = [initial_position]

    it = 0
    while it < n_samples:
        it += 1
        curr_theta = samples[-1].copy()
        idx = (it-1) % 4  # systematic

        if idx == 0:
            sigma_sq_proposal = sample_ssq_conditional_posterior(data, curr_theta)
            curr_theta[0] = sigma_sq_proposal

        elif idx == 1:
```

```
            tau_proposal = sample_tau_conditional_posterior(data, curr_theta)
            curr_theta[1] = tau_proposal

        elif idx == 2:
            mu_proposal = sample_mu_conditional_posterior(data, curr_theta)
            curr_theta[2:4] = mu_proposal

        elif idx == 3:
            gam_proposal = sample_gam_conditional_posterior(data, curr_theta)
            curr_theta[4:6] = gam_proposal

        else:
            print("Error: trying to update out-of-bounds parameter!")
        samples.append(curr_theta)

    return np.array(samples)
```

The Gibbs Sampling algorithm has a number of design choices, including (a) systematic vs random scan, and (b) block vs individual parameter updates. Note: there is no hyperparameter tuning for step size here because we take a completely new sample (from an adaptive proposal distribution) at each step.

(a) **Systematic vs random scan**: While random scan ensures detail balance of our markov chain, I preferred to use a systematic scan since it was more efficient, especially over smaller sampling experiments (guarantees an equal number of updates for each).

(b) **Block updates for mu and gamma**: Since $\mu_1, \mu_2$ and $\gamma_1, \gamma_2$ would always come from the same distribution I found it more efficient to update these together, drawing from a 2-dim multivariate normal rather than separately. This will not work well if the parameters within each block are uncorrelated but that should not be the case for this problem since the components of mu and gamma are closely related.

**Results**: Using the above design choices, we produce the following samples from our posterior (figure 5).

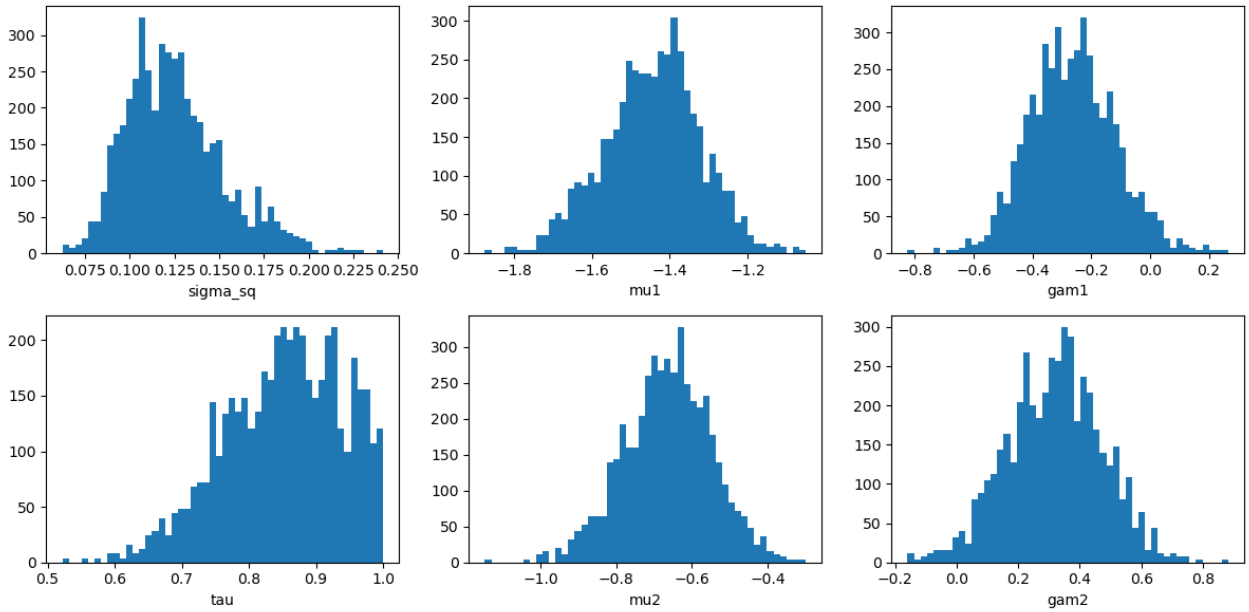


Figure 5: Histogram of Hamiltonian Monte Carlo posterior samples

| | sigma | tau | mu1 | mu2 | gam1 | gam2 |
|---|---|---|---|---|---|---|
| Mean of posterior samples | 0.13 | 0.85 | -1.45 | -0.66 | -0.26 | 0.32 |
| Variance of posterior samples | 0.001 | 0.008 | 0.016 | 0.014 | 0.022 | 0.024 |

The traceplots in figure 6 give an indication of how well the sampler explored the posterior density. I am comforted by the fact that there are no long periods of repeated values (getting stuck) despite us not implementing no-turn logic.

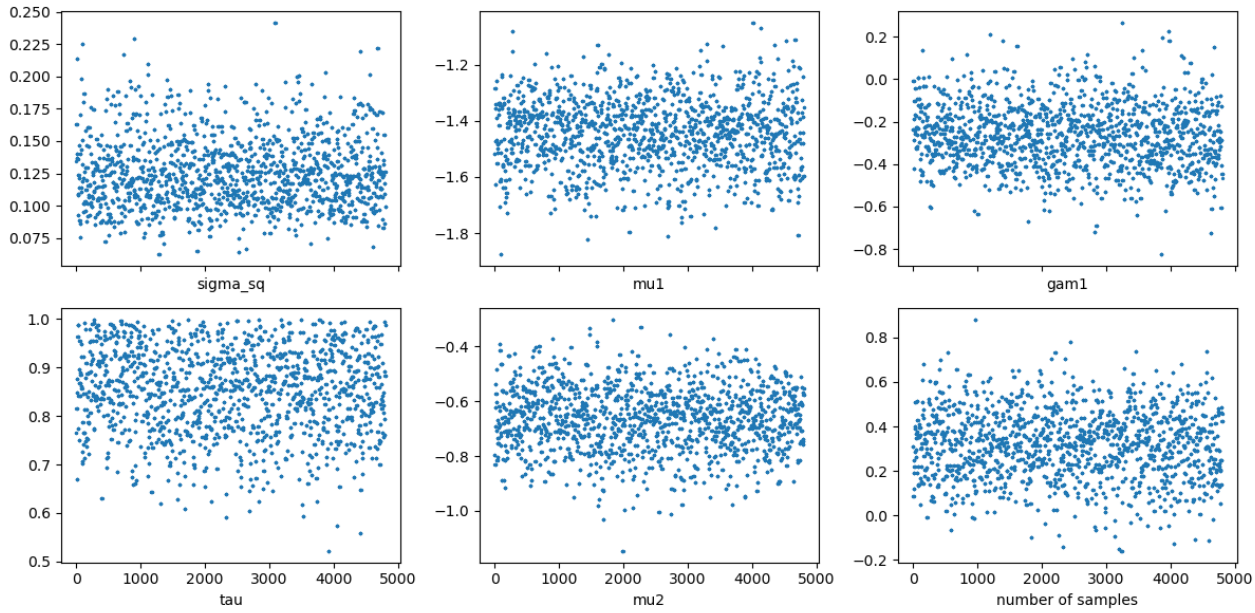


Figure 6: Traceplot for Hamiltonian Monte Carlo sampling

**Final remarks**. The Gibbs sampling algorithm is lightning fast compared to MH and HMC (25.0 sec!). This is because it does not execute an acceptance criterion! The main tradeoff is that our samples have significant autocorrelation (approx. 16 effective samples per 100 iterations). Note, the number of effective samples still surpasses MH but is almost 1/3 of our HMC result.

# 4 Importance Sampling

Importance Sampling is typically used to estimate summary statistics. For example, if $I_A(x)$ is the indicator function of x lying in some region A, and we know the target density $p(x)$, we can pick a proposal distribution $Q$ to sample from and weight our samples in the following way.

$$E[I_A(x)] = \int I_A(x)p(x)dx = \int I_A(x)\frac{p(x)}{q(x)}q(x)dx \approx \frac{1}{n}\sum I_A(x)\frac{p(x)}{q(x)}$$

Adapting this theory to our problem, we want to derive our target density $p(\theta|y)$ (likliehood x prior) and find a suitable proposal distribution Q that we can sample from to produce a set of samples and corresponding weights for each. Scaling each sample by its respective weight will give us posterior samples!

```
1    def multistep_importance_sampling(data, n_samples, initial_position):
2    samples, weights = [initial_position], [0]
```

```
    for it in range(n_samples):

        # sample from proposal dist q
        theta = sample_from_proposal_dist(data)
        samples.append(theta)

        # evaluate p(theta) and q(theta) to compute weight
        p_theta = joint_posterior_density(data, theta)  # trial density
        q_theta = compute_proposal_density(data, theta)  # proposal
        weights.append(p_theta/q_theta)

    return np.array(samples), np.array(weights)
```

The importance sampling method requires us to make a number of design choices, including (a) selecting a proposal distribution, (b) whether to use a normalized scheme, and (c) choice of prior for $\tau$.

(a) **Proposal distribution**. Want to find a proposal distribution that is as similar as possible (but with greater variance) to distribution described by our target density. Idea: split evaluation of the posterior into two separate steps:

- **Step 1**: Use only groups 1 and 2 to generate a closed form partial posterior ($\mu, \gamma, \sigma^2$ only). Combine with some $p(\tau)$ to generate our proposal distribution Q
- **Step 2**: Use output from step 1 as our new prior, then update using likelihood of data from groups 3 and 4 given parameters $\theta$ to generate our target density (equivalent to joint posterior)

This two-step approach is attractive because it uses our data effectively to inform our proposal distribution. We will also see that it simplifies our weight calculation. Let $\theta = (\mu, \gamma, \tau, \sigma^2)$.

$$\begin{aligned} p(\theta|Y_{1,2,3,4}) &\propto p(\mu, \gamma, \sigma^2|Y_{1,2}) \cdot p(\tau) \cdot L(Y_{3,4}|\theta) \\ &\propto p(\mu, \gamma, \sigma^2|Y_{1,2}) \cdot p(\tau) \cdot \prod_{t \in 3,4} p(y_i|\theta) \\ &\propto q(\theta) \cdot \prod_{t_i \in 3,4} p(y_i|\theta) \qquad \text{choose proposal } q(\theta) \end{aligned}$$

Want to find closed form distribution for $p(\mu, \gamma, \sigma^2|Y_{1,2})$. First recognize we can separate $\sigma^2$ from $\mu$ and $\gamma$.

$$p(\mu, \gamma, \sigma^2|Y_{1,2}) = p(\mu, \gamma|Y_{1,2}) \cdot p(\sigma^2|\mu, \gamma, Y_{1,2})$$

Derive $p(\sigma^2|\mu, \gamma, Y_{1,2})$: $\sigma^2$ conditional on $\mu$, $\gamma$ and groups 1, 2.

$$p(\sigma^2|\mu, \gamma, Y_{1,2}) \propto \frac{1}{\sigma^2} \cdot \left(\frac{1}{\sigma^2}\right)^{n_1+n_2} \exp\left(-\frac{1}{2\sigma^2}(\sum_{t_i=1} \|y_i - \mu\|_2^2 + \sum_{t_i=2} \|y_i - \gamma\|_2^2)\right)$$

$$\begin{aligned}
&\text{let } M_{\sigma^2} = \sum_{t_i=1} \|y_i - \mu\|_2^2 + \sum_{t_i=2} \|y_i - \gamma\|_2^2 \\
&\propto \left(\frac{1}{\sigma^2}\right)^{n_1+n_2+1} \exp\left(-\frac{M_{\sigma^2}}{2} \cdot \frac{1}{\sigma^2}\right) \\
&\propto \text{InvGamma}\left(n_1 + n_2, \quad \frac{M_{\sigma^2}}{2}\right)
\end{aligned}$$

Derive $p(\mu, \gamma | Y_{1,2})$: the marginal distribution of $\mu$ and $\gamma$.

$$\begin{aligned}
p(\mu, \gamma | Y_{1,2}) &= \int_0^\infty p(\mu, \gamma, \sigma^2 | Y_{1,2}) d\sigma^2 \\
&\propto \int_0^\infty \left(\frac{1}{\sigma^2}\right)^{n_1+n_2+1} \exp\left(-\frac{1}{2\sigma^2}(\sum_{t_i=1} \|y_i - \mu\|_2^2 + \sum_{t_i=2} \|y_i - \gamma\|_2^2)\right) d\sigma^2 \\
\text{recall} \quad \frac{\Gamma(\alpha)}{\beta^\alpha} &= \int_0^\infty (\sigma^2)^{-\alpha-1} \exp\left(\frac{\beta}{\sigma^2}\right) d\sigma^2 \\
&\propto \Gamma(n_1 + n_2) \left(-\frac{1}{2}(\sum_{t_i=1} \|y_i - \mu\|^2 + \sum_{t_i=2} \|y_i - \gamma\|^2)\right)^{-(n_1+n_2)} \\
&\propto (\sum_{t_i=1} \|y_i - \overline{Y_1}\|^2 + \sum_{t_i=2} \|y_i - \overline{Y_2}\|^2 + n_1 \|\overline{Y_1} - \mu\|^2 + n_2 \|\overline{Y_2} - \gamma\|^2)^{-(n_1+n_2)} \\
\text{define} \quad M_{\mu,\gamma} &= \sum_{t_i=1} \|y_i - \overline{Y_1}\|^2 + \sum_{t_i=2} \|y_i - \overline{Y_2}\|^2 \\
&\propto (M_{\mu,\gamma} + n_1 \|\overline{Y_1} - \mu\|^2 + n_2 \|\overline{Y_2} - \gamma\|^2)^{-(n_1+n_2)} \\
&\propto \left(M_{\mu,\gamma} + \left[\begin{pmatrix}\mu \\ \gamma\end{pmatrix} - \begin{pmatrix}\overline{Y_1} \\ \overline{Y_2}\end{pmatrix}\right]^T \begin{pmatrix} n_1 & 0 & 0 & 0 \\ 0 & n_1 & 0 & 0 \\ 0 & 0 & n_2 & 0 \\ 0 & 0 & 0 & n_2 \end{pmatrix} \left[\begin{pmatrix}\mu \\ \gamma\end{pmatrix} - \begin{pmatrix}\overline{Y_1} \\ \overline{Y_2}\end{pmatrix}\right]\right)^{-(n_1+n_2)} \\
&\propto \left(1 + \frac{2(n_1+n_2)-4}{M_{\mu,\gamma}(2(n_1+n_2)-4)} \left[\begin{pmatrix}\mu \\ \gamma\end{pmatrix} - \begin{pmatrix}\overline{Y_1} \\ \overline{Y_2}\end{pmatrix}\right]^T \begin{pmatrix} n_1{-}1 & 0 & 0 & 0 \\ 0 & n_1^{-1} & 0 & 0 \\ 0 & 0 & n_2{-}1 & 0 \\ 0 & 0 & 0 & n_2{-}1 \end{pmatrix}^{-1} \right. \\
&\qquad \left. \left[\begin{pmatrix}\mu \\ \gamma\end{pmatrix} - \begin{pmatrix}\overline{Y_1} \\ \overline{Y_2}\end{pmatrix}\right]\right)^{-(n_1+n_2)} \\
&\propto \left[1 + \frac{1}{\nu}(x-\eta)^T \Sigma^{-1} (x-\eta)\right]^{-\frac{\nu+4}{2}} \quad \text{recognize t-distribution density} \\
\text{where} \quad \nu = 2(n_1+n_2)-4, \Sigma &= \frac{M_{\mu,\gamma}}{2(n_1+n_2)-4} \begin{pmatrix} n_1{-}1 & 0 & 0 & 0 \\ 0 & n_1^{-1} & 0 & 0 \\ 0 & 0 & n_2{-}1 & 0 \\ 0 & 0 & 0 & n_2{-}1 \end{pmatrix}, \eta = \begin{pmatrix}\overline{Y_1} \\ \overline{Y_2}\end{pmatrix}
\end{aligned}$$

$$\therefore p(\mu, \gamma \mid Y_{1,2}) \sim t_4 \left( \begin{pmatrix} \overline{Y_1} \\ \overline{Y_2} \end{pmatrix}, \frac{M_{\mu,\gamma}}{2(n_1+n_2)-4} \begin{pmatrix} n_1^{-1} & 0 & 0 & 0 \\ 0 & n_1^{-1} & 0 & 0 \\ 0 & 0 & n_2^{-1} & 0 \\ 0 & 0 & 0 & n_2^{-1} \end{pmatrix} \right)$$

Putting these parts together we have the following proposal distribution,

$$q(\theta) \sim t_4 \left( \begin{pmatrix} \overline{Y_1} \\ \overline{Y_2} \end{pmatrix}, \frac{M_{\mu,\gamma}}{2(n_1+n_2)-4} \begin{pmatrix} n_1^{-1} & 0 & 0 & 0 \\ 0 & n_1^{-1} & 0 & 0 \\ 0 & 0 & n_2^{-1} & 0 \\ 0 & 0 & 0 & n_2^{-1} \end{pmatrix} \right) \cdot \text{InvGamma}\left( n_1+n_2, \quad \frac{M_{\sigma^2}}{2} \right)$$

(b) **Normalized importance sampling**. Note that we cannot solve for closed form distribution of our joint posterior after groups 1 and 2. Idea: let our target $p(x) = f(x)/z_p$ and $q(x) = g(x)/z_q$ where we can evaluate $f(x), g(x)$ but not $p(x), q(x)$ are $z_p, z_q$ are constants.

If we draw $x_j$ $j = 1, 2, ...n$ iid samples from Q distribution, then compute weights $u_j = f(x_j)/g(x_j)$ and apply these to our samples, we can recover the weights as if we had been able to evaluate $p(x)$ and directly compute $w_j = p(x_j)/q(x_j)$:

$$\hat{x_j} = \frac{h(x_j) \cdot w_j}{\sum_{i=1}^n w_j} = \frac{\frac{z_p}{z_q} \cdot h(x_j) \cdot \frac{f(x_j)}{g(x_j)}}{\sum_{i=1}^n \frac{f(x_j)}{g(x_j)}} = \frac{h(x_j) \cdot u_j}{\sum_{j=1}^n u_j}$$

This is important for our particular problem since it is not possible to compute the closed form posterior (target density) and so instead we need to make do with some $f(x) \propto p(x)$, which we take to be the likelihood x prior.

Note our choice of $q(\theta)$ simplies the weight calculation to just evaluating the likelihood of groups 3 and 4 given the samples $\theta_j$ from proposal distribution $q$.

$$u_j = \frac{f(\theta_j)}{g(\theta_j)} = \frac{p(\theta_j|Y_{1,2,3,4})}{p(\mu_j, \gamma_j, \sigma_j^2|Y_{1,2}) \cdot p(\tau_j)} = \frac{q(\theta_j) \cdot \prod_{t_i \in 3,4} p(y_i|\theta_j)}{q(\theta_j)} = \prod_{t_i \in 3,4} p(y_i|\theta_j)$$

(c) **Choice of tau prior**. While we were able to solve for a closed form of the posterior for groups 1 and 2, this does not give us any additional information about $\tau$ to use as a prior for Step 2. I tried both using a beta centered at zero Beta$(5, 5)$ and a beta centered at 0.9 Beta$(20, 3)$ (closer to the MLE estimate). As expected the latter worked much better since it generated more samples closer to our target.

**Results**: Using the above design choices we want to generate the posterior marginals for our parameters $\theta$. Unlike for our other sampling methods, we do not get these as a direct output from the algorithm. Idea: specify a set of binned regions $(A)$ for each parameter that cover

the region of posterior marginal density, then use our samples $\theta_j$ from proposal distribution $q$ and corresponding weights $w_j$ to compute $\mathbb{E}[I_A(\theta)]$ for each region $A$. These "expectations of indicator functions" are the probabilities of a given parameter falling in region $A$, which we can now plot in a histogram as individual bars in our marginal posterior histogram.

$$\mathbb{E}_{\theta\sim|}[I_A(\theta)] = \frac{1}{n}\sum_j I_A(\theta_i)\frac{p(\theta_i)}{q(\theta_i)} = \frac{1}{n}\sum_i I_A(\theta_i)\frac{f(\theta_i)}{g(\theta_i)} = \frac{1}{n}\sum_i I_A(\theta_i)u_i$$

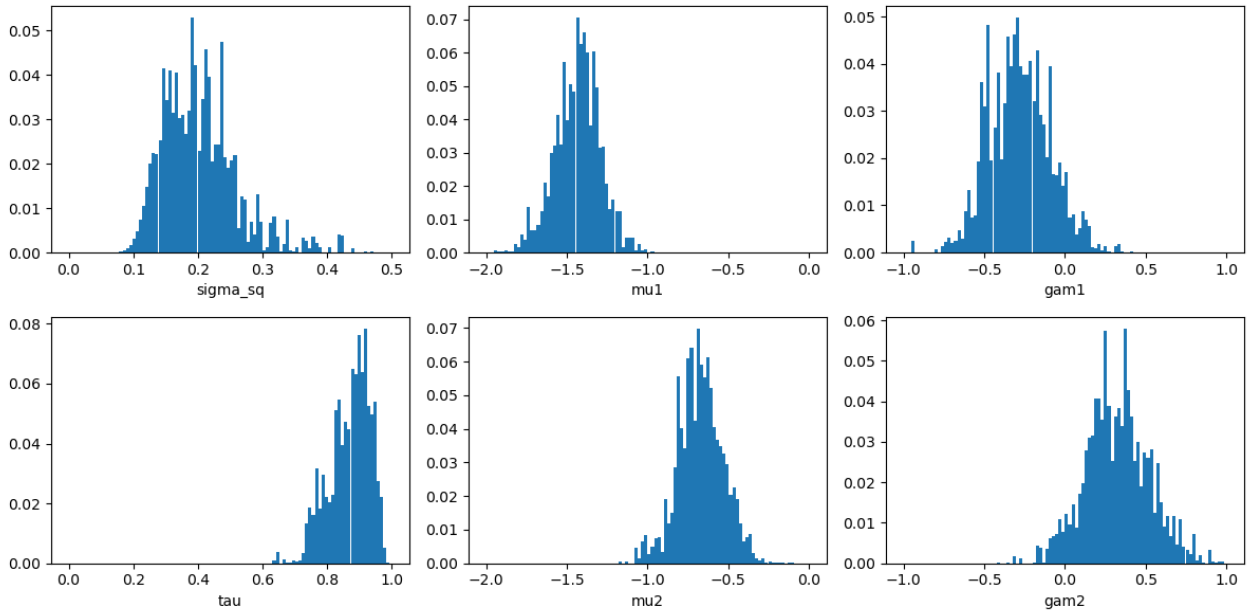


Figure 7: Histogram of Importance Sampling posterior samples

| | sigma | tau | mu1 | mu2 | gam1 | gam2 |
|---|---|---|---|---|---|---|
| Mean of weighted samples | 0.20 | 0.88 | -1.42 | -0.66 | -0.27 | 0.34 |
| Variance of weighted samples | 0.022 | 0.004 | 0.020 | 0.077 | 0.044 | 0.058 |

**Final remarks**. While the Importance Sampling algorithm could be highly parallelized (each sample is independent) we needed to run many more samples than other methods (1,000,000 vs 5,000) to sufficiently cover our desire regions for the marginal posterior plot. Moreover, the algorithm was VERY senstivie to the closeness of our proposal distribution $q$. In this case we were able to compute a good closed form approximation from groups 1 and 2, but one could imagine this not generalizing well to problems with more data or more complex parameterizations.