AE567 Project 2 Part 2: Bayesian Inference for Nonlinear Models

Akshat Dubey

1 Delayed Rejection Adaptive Metropolis on the Banana Distribution

First lets take a look a the target distribution that we would like to sample from, which is the banana distribution. In our case the distribution is defined as

$$(x_1, x_2 + (x_1^2 + 1)) \sim \mathcal{N}(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1 & 0.9\\0.9 & 1\end{bmatrix})$$

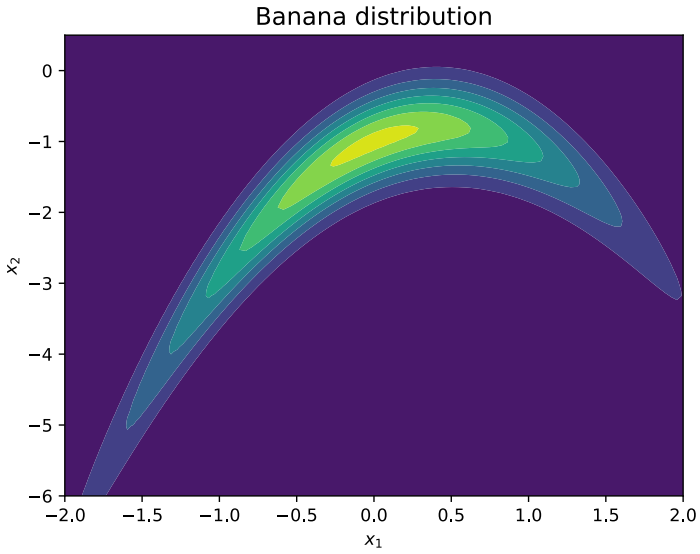

Figure 1: PDF of Banana distribution

Since for all of our use cases, we will be using the log PDF of the distribution, we need to derive it. For a standard multivariate gaussian ($x \sim \mathcal{N}(x|\mu, \Sigma)$), the PDF is given by (ref: wikipedia)

Let

d: dimension of x
μ: mean of x
Σ: covariance matrix of x
$|\Sigma|$: determinant of Σ

Then,

$$f_X(x) = \frac{\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)}{\sqrt{(2\pi)^d|\Sigma|}}$$

Hence the logPDF is given by

$$log(f_X) = -\frac{(x-\mu)^T\Sigma^{-1}(x-\mu)}{2} - \frac{d.ln(2\pi)}{2} - \frac{ln|\Sigma|}{2}$$

substituting the values of μ, Σ for the banana distribution

$$log(f_X) = -\frac{1}{2}x^T \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}^{-1} x - ln(2\pi) - \frac{ln(0.19)}{2}$$

Everything here except for x^T and x is a constant, hence we can precompute and store them to improve computational efficiency.

We can take any inputs (x_1, x_2), transform them to $(x_1, x_2 + (x_1^2 + 1))$ and then plug that into the logPDF to get the logPDF of the banana distribution.

Before we move on to Delayed Rejection Adaptive Metropolis(DRAM), lets first set a baseline by implementing the standard Metropolis Hastings(MH) Markov Chain Monte Carlo(MCMC) algorithm. The Metropolis Hastings algorithm is a simple MCMC algorithm that generates a Markov chain that should converge to the banana distribution. The algorithm is as follows:

Let

$q(y|x)$: proposal distribution that proposes y given x
$f_{Xb}(x)$: logPDF of x in the banana distribution
$f_{Xq}(x|y)$: logPDF of x given y in the proposal distribution
N: Number of samples requested from the Markov Chain

1. Set initial state $x^{(0)}$ and proposal distribution $q(y|x^{(k)})$
2. For k = 1 to N:
    1. Generate a proposal $y \sim q(y|x^{(k)})$
    2. Calculate the acceptance ratio $a_1(x^{(k)}, y) = min(1, \exp(f_{Xb}(y) - f_{Xb}(x^{(k)}) + f_{Xq}(x^{(k)}|y) - f_{Xq}(y|x^{(k)})))$
    3. Set $x^{(k+1)}$ as follows:
    $$x^{(k+1)} = \begin{cases} y & \text{with probability } a_1 \\ x^{(k)} & \text{otherwise} \end{cases}.$$
    4. Increment k
3. Return the samples $x^{(1)}, x^{(2)}, \ldots, x^{(N)}$ as the Markov chain

Note that since we are using a random walk proposal, the probability of proposing y given $x^{(k)}$ is the same as the probability of proposing $x^{(k)}$ given y, hence, the terms $f_{Xq}(x^{(k)}|y)$ and $f_{Xq}(y|x^{(k)})$ cancel out in the acceptance ratio. and the acceptance ratio simplifies to $a_1(x, y) = min(1, \exp(f_{Xb}(y) - f_{Xb}(x^{(k)}))$. Since we will be using the same proposal distribution for all iterations, we can ignore the proposal distribution terms in the acceptance ratio a_1 from now on.

To get started with the Metropolis Hastings algorithm, we need to find a way to get the initial state and a proposal distribution as needed in step 1. To get the initial state, we can start off from the Maximum Aposteriori(MAP) estimate of the banana distribution. The MAP estimate can be found by maximizing the logPDF of the banana distribution. If we simplify the problem and approximate the distribution as a 2D gaussian, the scaled negative inverse of the hessian at the MAP point can be used as the covariance matrix for a random walk proposal. This is also called the Laplace Approximation. The scaling factor of the negative inverse hessian is important for determining the covariance of the the random walk proposal and has been taken from the suggestion in the notes (ref: eq 16.6). Specifically:

$$\begin{aligned} x_{MAP} &= \arg\max_x f_{Xb}(x) \\ \Sigma_{proposal} &= -\frac{2.4^2}{d}\left(\nabla^2 f_{Xb}(x_{MAP})\right)^{-1} \\ x^{(0)} &= x_{MAP} \\ q(y|x) &= \mathcal{N}(y|x, \Sigma_{proposal}) \end{aligned}$$

Hence for the random walk proposal, we can sample it using the Cholesky decomposition of the covariance matrix $\Sigma_{proposal}$ and adding it to the current state $x^{(k)}$ (ref: notes section 6.2) just like project 2.a. We can also calculate its logPDF the same way we did for the banana distribution without transforming the inputs (x_1, x_2) to $(x_1, x_2 + (x_1^2 + 1))$.

Using `scipy.optimize.minimize` with $-f_{Xb}$ as the objective function, we find the MAP and proposal covariance matrix:

- x_{MAP}: (-9.44e-08, -1)
- $$\Sigma_{proposal} : \begin{bmatrix} 2.88 & 2.59 \\ 2.59 & 2.88 \end{bmatrix}$$

Having established a baseline of MH with a starting $x^{(0)}$ and a proposal distribution $q(y|x)$, we can now move on to other MCMC algorithms.

1.1 Adaptive Metropolis

The Adaptive Metropolis (AM) algorithm consists of the same basic steps as the MH algorithm, but instead of using the same proposal distribution for all iterations, it adapts the proposal distribution based on the samples generated so far. Once a specified threshold of iterations has been reached, it starts using the adapted covariance instead of $\Sigma_{proposal}$ The algorithm is as follows:

1. Set initial state $x^{(0)}$ and proposal distribution $q(y|x^{(k)})$
2. For k = 1 to N:
    1. Generate a proposal $y \sim q(y|x^{(k)})$
        - If k $\leq$ threshold, proposal covariance = $\Sigma_{proposal}$
        - Otherwise, proposal covariance = covariance of all samples generated so far
    2. Calculate the acceptance ratio $a_1(x, y) = min(1, \exp(f_{Xb}(y) - f_{Xb}(x^{(k)})))$
    3. Set $x^{(k+1)}$ as follows:
    $$x^{(k+1)} = \begin{cases} y & \text{with probability } a_1 \\ x^{(k)} & \text{otherwise} \end{cases} .$$
    4. **Update the proposal distribution $q(y|x^{(k)})$ based on the samples generated so far**
    5. Increment k

Notice that the only additional step here compared to MH is step 4, where the proposal distribution is updated based on the samples generated so far. The proposal distribution is updated by taking the empirical covariance of the samples generated so far and using it as the covariance matrix of the proposal distribution. When generating a lot of samples, taking the empirical covariance of all the previous samples is computationally expensive and actually wasteful since we perform redundant calculations that we have already done in previous iterations. Hence the covariance was updated recursively using the sample mean which was also updated recursively. The recursive update of the mean and covariance is as follows(ref: eq 16.9 and 16.10 from notes):

$$\begin{aligned} &\text{Let} \\ \bar{x}^{(k)}&: \text{ mean of all samples up to } k \\ S^{(k)}&: \text{ covariance of all samples up to } k \\ s_d = \frac{2.4^2}{d}&: \text{ scaling factor for proposal covariance} \\ \xi = 1e-7&: \text{ constant to keep the covariance matrix well-conditioned} \\ \bar{x}^{(k)} &= \frac{1}{k+1}x^{(k)} + \frac{k}{k+1}\bar{x}^{(k-1)} \\ S^{(k+1)} &= \frac{k-1}{k}S^{(k)} + \frac{s_d}{k}(\xi I + k\bar{x}^{(k-1)}\bar{x}^{(k-1)^T} - (k+1)\bar{x}^{(k)}\bar{x}^{(k)^T} + x^{(k)}x^{(k)^T}) \end{aligned}$$

We can start off this process with

$$\begin{aligned} \bar{x}^{(0)} &= x^{(0)} \\ S^{(0)} &= \Sigma_{proposal} \end{aligned}$$

This way, we can adapt the proposal distribution based on the samples generated so far, so that $q(y|x^{(k)}) \sim \mathcal{N}(x^{(k)}, S^{(k)})$ and hopefully get the acceptance ratio closer to 20%-30% and samples closer to the banana distribution.

Take note that the scaling factor s_d need not be the same as defined above and can be used as a tuning knob to adjust the acceptance ratio. Increasing it will cause the proposal distribution to be more exploratory, reducing the acceptance ratio and vice versa. It may seem that the adaptation threshold is also a tuning knob, but I did not explore it in this project because setting it too high did not make sense to me as we want to adapt as soon as possible. I reject the first half of the samples generated anyway, so even if the initial samples have a bad estimate of the actual covariance, they will not be taken into account. In addition, the covariance matrix is updated every iteration instead of at a set frequency so that it can stay as close to the true covariance as possible unlike in the notebook we discussed in class.

1.2 Delayed Rejection

The Delayed Rectection (DR) algorithm is a variant of the MH algorithm that tries to improve the acceptance ratio by using multiple proposals in a single iteration. For this project we will only use one secondary proposal which will use the covariance matrix of the first proposal scaled by a constant γ. To preserve the reversibility of the markov chain, the acceptance ratio a_2 is calculated differently from a_1. In addition, the second proposal y_2 will only depend on the current state $x^{(k)}$ and not on the rejected initial proposal y_1, simplifying the calculation of the acceptance ratio.

The acceptance ratio a_2 is calculated as follows:

Let

$$
\begin{aligned}
y_1&: \text{ first proposal sample}\\
y_2&: \text{ second proposal sample}\\
\gamma&: \text{ scaling factor for the second proposal covariance}\\
q_1(y_1|x^{(k)}) = \mathcal{N}(y_1|x^{(k)}, \Sigma_{proposal})&: \text{ first proposal distribution}\\
q_2(y_2|x^{(k)}) = \mathcal{N}(y_2|x^{(k)}, \gamma\Sigma_{proposal})&: \text{ second proposal distribution}\\
f_{Xq_1}(x|y)&: \text{ logPDF of x given y in the first proposal distribution}\\
f_{Xq_2}(x|y)&: \text{ logPDF of x given y in the second proposal distribution}
\end{aligned}
$$

Then

$$
\begin{aligned}
a_2(x^{(k)}, y_1, y_2) = min(1, \exp(&f_{Xb}(y_2) - f_{Xb}(x^{(k)}) + f_{Xq_1}(y_1|y_2) - f_{Xq_1}(y_1|x^{(k)})\\
&+ f_{Xq_2}(x|y_2, y_1) - f_{Xq_2}(y_2|y_1, x^{(k)}) + log(1 - a_1(y_2, y_1)) - log(1 - a_1(x^{(k)}, y_1)))
\end{aligned}
$$

Since our second proposal only depends on $x^{(k)}$

$$
\begin{aligned}
a_2(x^{(k)}, y_1, y_2) = min(1, \exp(&f_{Xb}(y_2) - f_{Xb}(x^{(k)}) + f_{Xq_1}(y_1|y_2) - f_{Xq_1}(y_1|x^{(k)})\\
&+ f_{Xq_2}(x|y_2) - f_{Xq_2}(y_2|x^{(k)}) + log(1 - a_1(y_2, y_1)) - log(1 - a_1(x^{(k)}, y_1)))
\end{aligned}
$$

Since the proposal q_2 is symmetric, $f_{Xq_2}(x^{(k)}|y_2) = f_{Xq_2}(y_2|x^{(k)})$, so they cancel out.

$$
a_2(x^{(k)}, y_1, y_2) = min(1, \exp(f_{Xb}(y_2) - f_{Xb}(x^{(k)}) + f_{Xq_1}(y_1|y_2) - f_{Xq_1}(y_1|x^{(k)}) + log(1 - a_1(y_2, y_1)) - log(1 - a_1(x^{(k)}, y_1)))
$$

The algorithm is as follows:

1. Set initial state $x^{(0)}$ and proposal distribution $q_1(y|x^{(k)})$
2. For k = 1 to N:
    1. Generate a proposal $y_1 \sim q_1(y_1|x^{(k)})$
    2. Calculate the acceptance ratio $a_1(x^{(k)}, y) = min(1, \exp(f_{Xb}(y) - f_{Xb}(x^{(k)})))$
    3. Accept $x^{(k+1)} = y_1$ with probability a_1

- If accepted, skip to step 4
- if rejected:
    1. Generate a second proposal $y_2 \sim q_2(y_2|x^{(k)})$
    2. Calculate the acceptance ratio $a_2(x^{(k)}, y_1, y_2)$ using the formula above
    3. Set $x^{(k+1)}$ as follows:
$$x^{(k+1)} = \begin{cases} y_2 & \text{with probability } a_2 \\ x^{(k)} & \text{otherwise} \end{cases}.$$
4. Increment k

Here the only tuning knob we have is γ, which can be used to adjust the acceptance ratio. Increasing it will cause the proposal distribution to be more exploratory, reducing the acceptance ratio and vice versa. Since it is used when the first proposal is too exploratory and rejected, it should be ≤ 1 to ensure that the second proposal is less exploratory and gets accepted.

1.3 Delayed Rejection Adaptive Metropolis

The Delayed Rejection Adaptive Metropolis (DRAM) algorithm is a combination of the AM and DR algorithms. It adapts the proposal distribution based on the samples generated so far and uses 2 levels of proposals in each iteration. Take note after the adaptation threshold is reached, the second proposal scales the estimated covariance of the samples so far and not $\Sigma_{proposal}$ The algorithm is as follows:

1. Set initial state $x^{(0)}$ and proposal distribution $q_1(y|x^{(k)})$
2. For k = 1 to N:
    1. Generate a proposal $y_1 \sim q_1(y_1|x^{(k)})$
        - If k $\leq$ threshold, proposal covariance = $\Sigma_{proposal}$
        - Otherwise, proposal covariance = covariance of all samples generated so far
    2. Calculate the acceptance ratio $a_1(x^{(k)}, y) = min(1, \exp(f_{Xb}(y) - f_{Xb}(x^{(k)})))$
    3. Accept $x^{(k+1)} = y_1$ with probability a_1
    - If accepted, skip to step 4
    - if rejected:
        1. Generate a second proposal $y_2 \sim q_2(y_2|x^{(k)})$ by scaling the covariance matrix of the first proposal by a constant γ like in DR
        2. Calculate the acceptance ratio $a_2(x^{(k)}, y_1, y_2)$ using the formula in DR
        3. Set $x^{(k+1)}$ as follows:
$$x^{(k+1)} = \begin{cases} y_2 & \text{with probability } a_2 \\ x^{(k)} & \text{otherwise} \end{cases}.$$
    4. Update the proposal distribution $q_1(y|x^{(k)})$ based on the samples generated so far using the formula in AM
    5. Increment k

Here we have 2 tuning knobs, γ and s_d, which can be used to adjust the acceptance ratio. Increasing γ will cause the proposal distribution to be more exploratory, reducing the acceptance ratio and vice versa. Increasing s_d will cause the proposal distribution to be more exploratory, reducing the acceptance ratio and vice versa. Since γ is used when the first proposal is too exploratory and rejected, it should be ≤ 1 to ensure that the second proposal is less exploratory and gets accepted. Again, even though it is available, i have not used the adaptation threshold as a tuning for the same reasons described in the AM section.

1.4 Banana Distribution (Already covered in section 1)

1.5 Results

N = 50000 samples were generated for each algorithm, out of which 50% were discarded and the remaining 50% were downsampled by another 50%, resulting in 12500 samples after burn-in. For the entire results section, all algorithms except MH have been tuned to result in an acceptance ratio of around 20%-30% while still maintaining good autocorrelation, marginals and looking good under visual inspection.

1.5.a Plots of 1D and 2D marginals

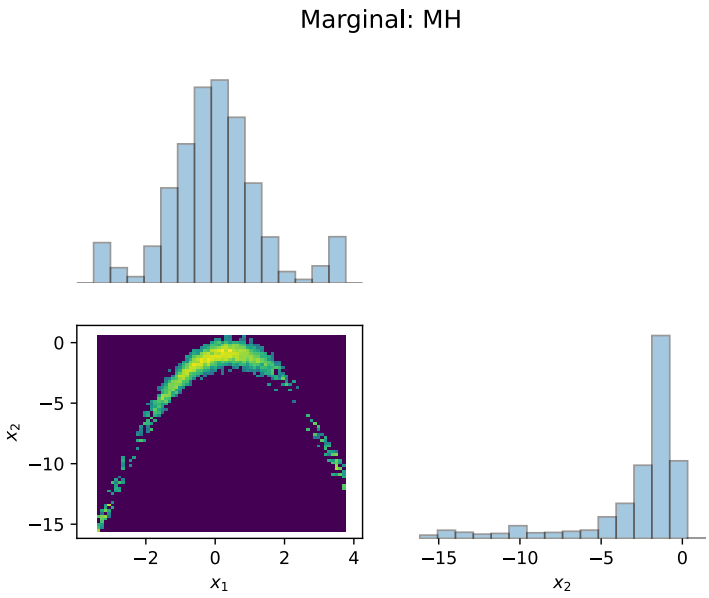

Figure 2: Marginals: Metropolis Hastings with seed = 3

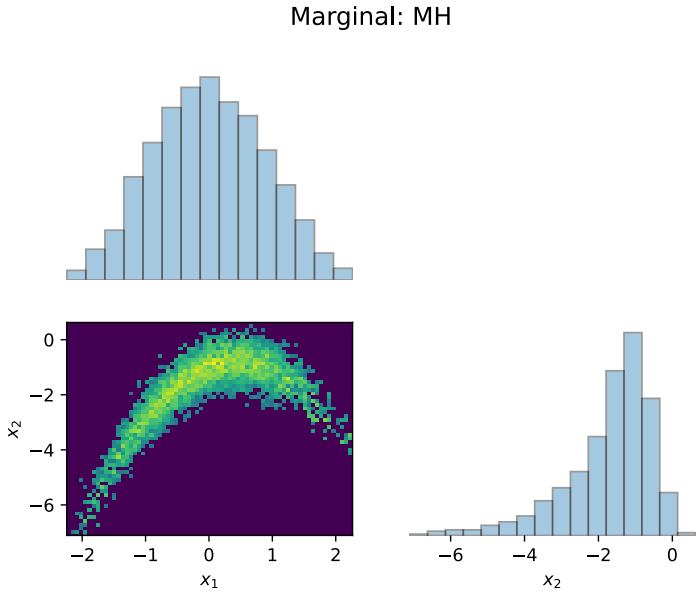

Figure 3: Marginals: Metropolis Hastings with seed = 0

The standard MH Algorithm was pretty inconsistent across runs, and I have displayed two runs of the same code but with different seeds for `np.random` and the results vary quite a bit. For the rest of the report, I am taking the worst case run of the MH algorithm so it is easier to see the benefits of the other algorithms over it. This inconsistency was not observed in the other algorithms, which prove why they are better than the standard MH algorithm.

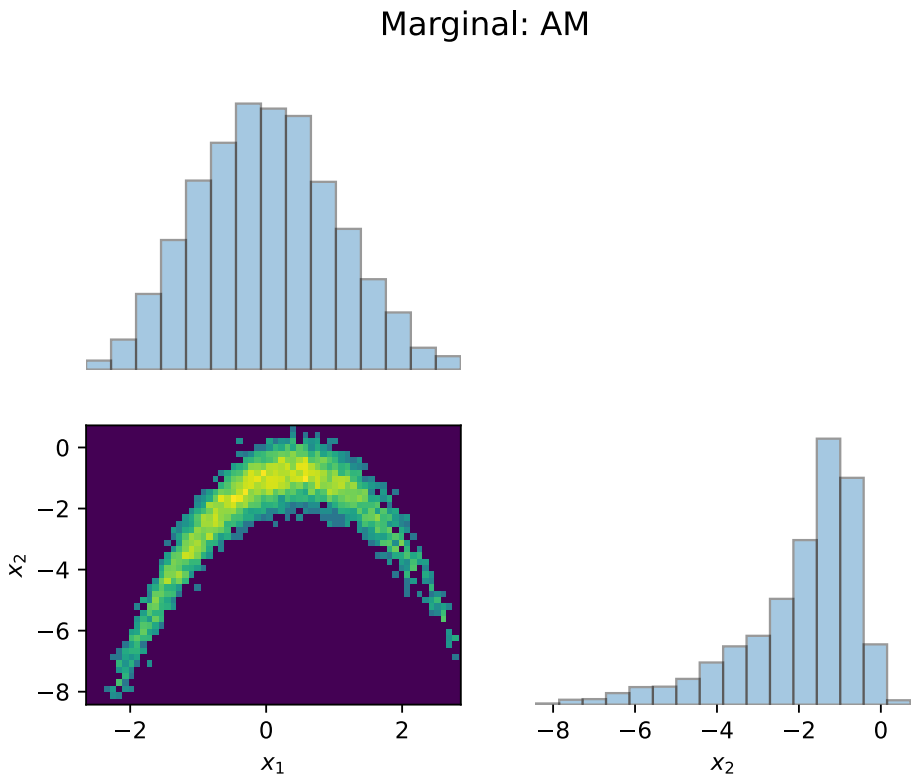

Figure 4: Marginals: Adaptive Metropolis

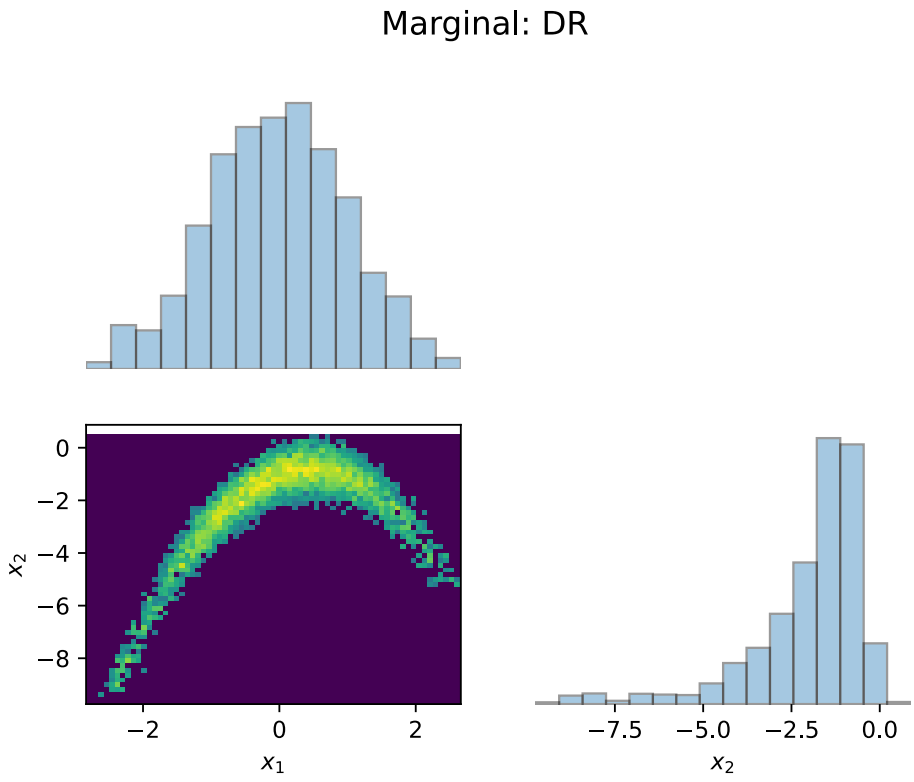

Figure 5: Marginals: Delayed Rejection

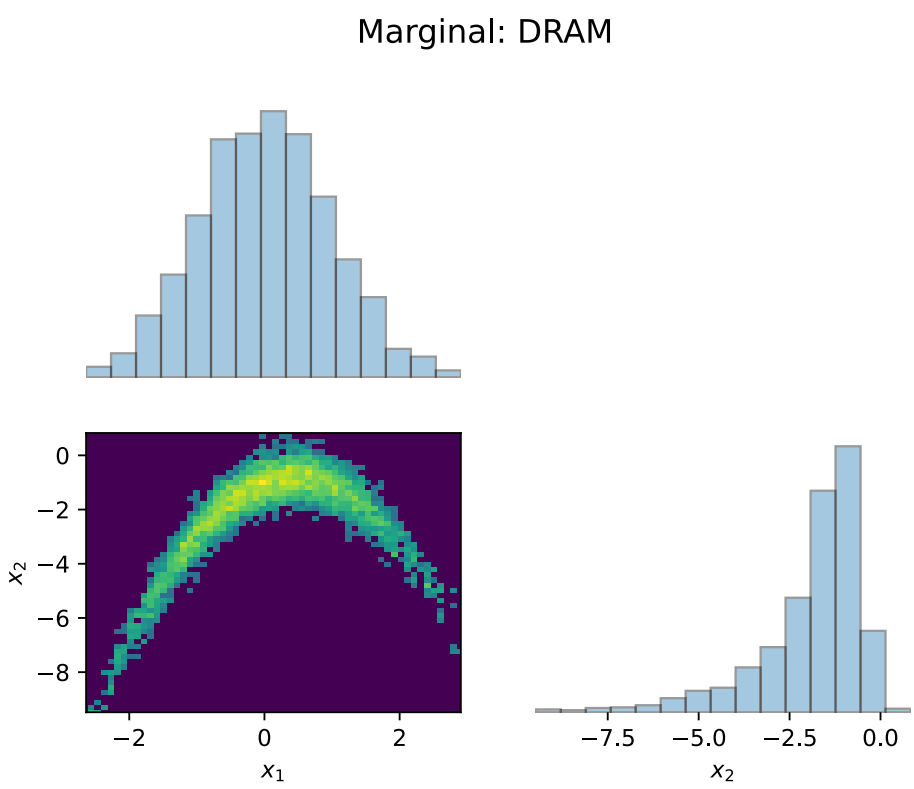

Figure 6: Marginals: Delayed Rejection Adaptive Metropolis

Here we expect the join distribution to look like the target banana distribution, and the marginals to represent the individual dimensions. Looking at the distributions, we can see that DRAM performs the best and MH performs the worst in terms of matching the original banana distribution. However, DRAM, AM and DR all have a similar performance in terms of matching the banana distribution and look pretty good.

1.5.b/1.5.c Autocorrelation plots and Integrated autocorrelation values

Autocorrelation represents the correlation between the samples at different k values, which are represented as lags. In an ideal Markov Chain, the samples should be uncorrelated except for with the previous sample since it is used to generate it. Realistically, is is impossible for the autocorrelation to drop to 0 after a lag of 1, so we expect the better performing algorithms to have very little autocorrelation beyond a small lag value.

The autocorrelation plots are plotted to a lag where the autocorrelation first reaches 0 for the worst algorithm. This is done to keep the x axis the same for easier comparison. We can observe that MH performs the worst as expected, and DRAM performs the best. Again, as we saw with the marginal plots, DRAM, AM and DR have similar performance in terms of autocorrelation. It is interesting to see that DR performs a lot worse in the x_2 dimension compared to x_1, which makes sense since the banana distribution resembles a gaussian less in the x_2 dimension. Since DR is not able to adapt its proposal distribution, it is not able to sample from the distribution in x_2 as well as the adaptive algorithms.

The Integrated Autocorrelation(IAC) values are calculated by summing the autocorrelation values for each axis up to a certain lag where the autocorrelation reaches 0. This is done to get an idea of how much total autocorrelation was present in the samples generated by each algorithm. The lower the IAC, the better the algorithm. Just like the autocorrelation plots, these values have been calculated to the lag where the autocorrelation first reaches 0 for the worst algorithm to have a fair comparison.

Integrated autocorrelation values for each axis:

- MH: [279.31530273 496.92620102]
- AM: [40.49130493 24.24136262]
- DR: [25.38595114 35.05717595]
- DRAM: [6.52753247 10.75545756]

Here, the difference between the algorithms is clearer. We can see that MH performs the worst by a huge margin, and DRAM performs the best, followed by AM and DR. An interesting artefact of using the same lag value to calculate the IAC is that for the better performing algorithms, the IAC actually has a lot of noise as can be seen in the Autocorrelation plots after the Autocorrelation reaches 0 for them. This might cause the IAC to be higher than it actually is. This happens because the better performing algorithms have a very small IAC to begin with and adding even a little noise inflates their score significantly. However it is still a good metric to compare the algorithms.

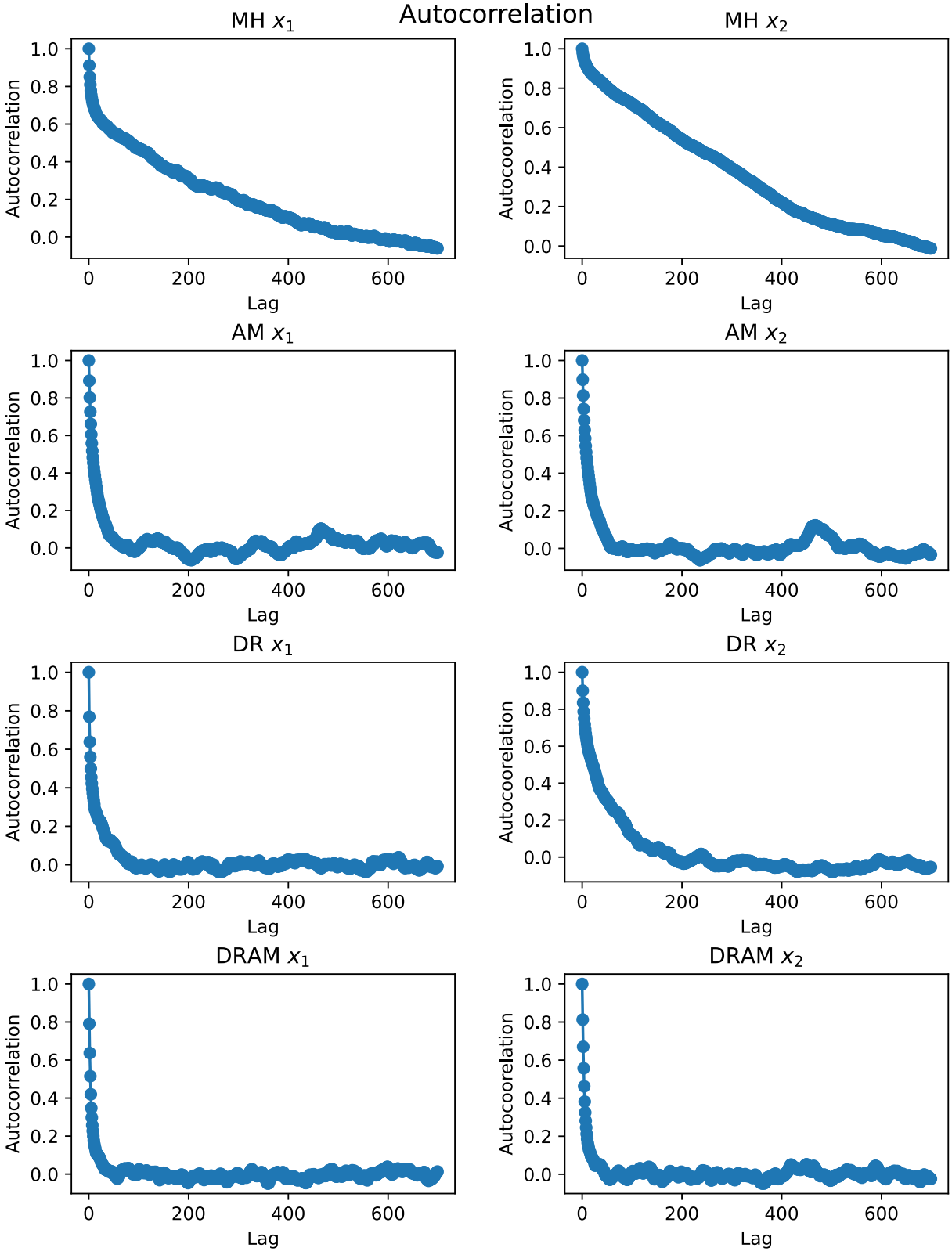

Figure 7: Autocorrelation

1.5.d Acceptance ratio

Acceptance ratio is the percentage of proposed samples that were accepted while running the algorithm. A good acceptance ratio is around 20%-30% as per the notebook we went through in class. A lower acceptance ratio means that the proposal distribution is too exploratory and the samples are being rejected too often. A higher acceptance ratio means that the proposal distribution is not exploratory enough and the samples are being accepted too often, which might cause the algorithm to get stuck in a small local area.

Acceptance ratios without tuning:

- MH: 22%
- AM: 14%
  - s_d: 2.88
- DR: 41%
  - γ: 0.5
- DRAM: 28%
  - γ: 0.5
  - s_d: 2.88

Acceptance ratios with tuning:

- MH: 22%
- AM: 26%
  - s_d: 0.9
- DR: 31%
  - γ: 2
- DRAM: 25%
  - γ: 0.7
  - s_d: 2.88

While tuning DR, I noticed that if i tried to push the acceptance ratio down by increasing γ beyond 2, the autocorrelation would become a lot worse, causing a much slower dropoff in autocorrelation and a higher IAC. Increasing γ beyond 1 was done just to explore its effects, knowing that it is not intended to be used like this. according Hence, for the sake of experimentation, I decided to keep it at 2 since it was giving a better result than when it was ≤ 1. For AM, I tried to push the acceptance ratio up by decreasing s_d, which worked as expected without significantly impacting the rest of the metrics. For DRAM, the acceptance ratio was already pretty good, so I only made a small adjustment to γ to get it to 25%.

1.5.e Visual inspection of mixing

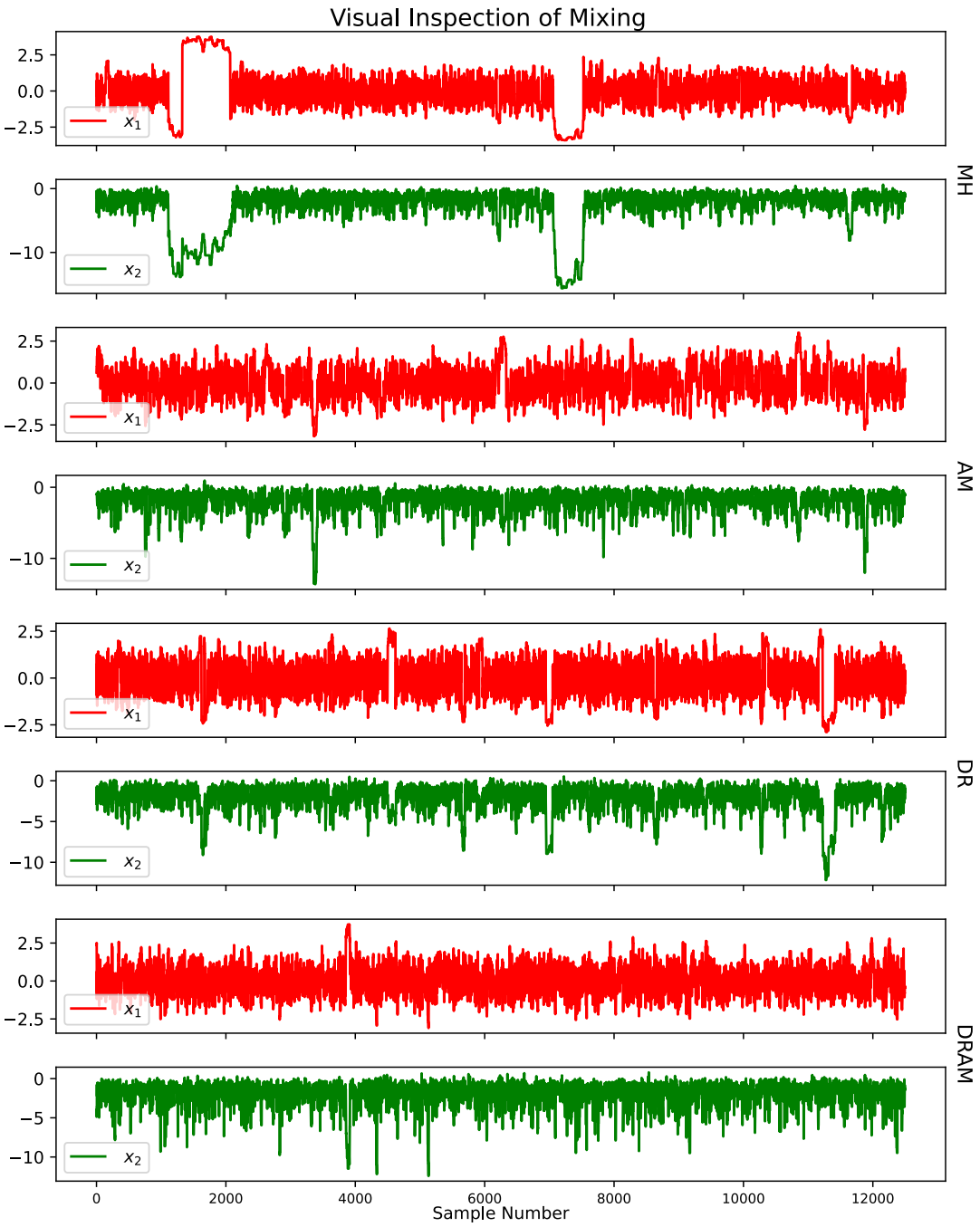

Figure 8: Visual Inspection of mixing

This visual inspection helps to see how truly random and uncorrelated the samples from each algorithm are. Here we expect to see a random distribution of points that are not correlated with each other. This means we should not see any trends in the samples. If we see that the samples are forming a pattern near a particular value, it indicates that the MC is stuck in a local area and is not sampling the whole distribution. As seen in previous metrics, we can see that DRAM performs the best, followed by AM and DR. MH performs the worst, as expected. MH has a lot of correlated samples near sample number 3500 and 7500, which is bad. DR also has some samples clumped together near sample number 4500, 7000 and 11000, but they do not stay together nearly as much as MH. AM also has three such clusters, near sample number 3500, 6500 and 11000, but they do not stay clustered together as much as DR. DRAM has the least amount of clustering, with only one small cluster near sample number 3800, which is not as bad as the other algorithms. Hence we can see that DRAM is performing the best.

2 Apply DRAM for Bayesian Inference

2.A Satellite Dynamics

After setting up the Satellite Dynamics, we can observe the true trajectory of the satellite and the noisy trajectory observed by the sensor data.

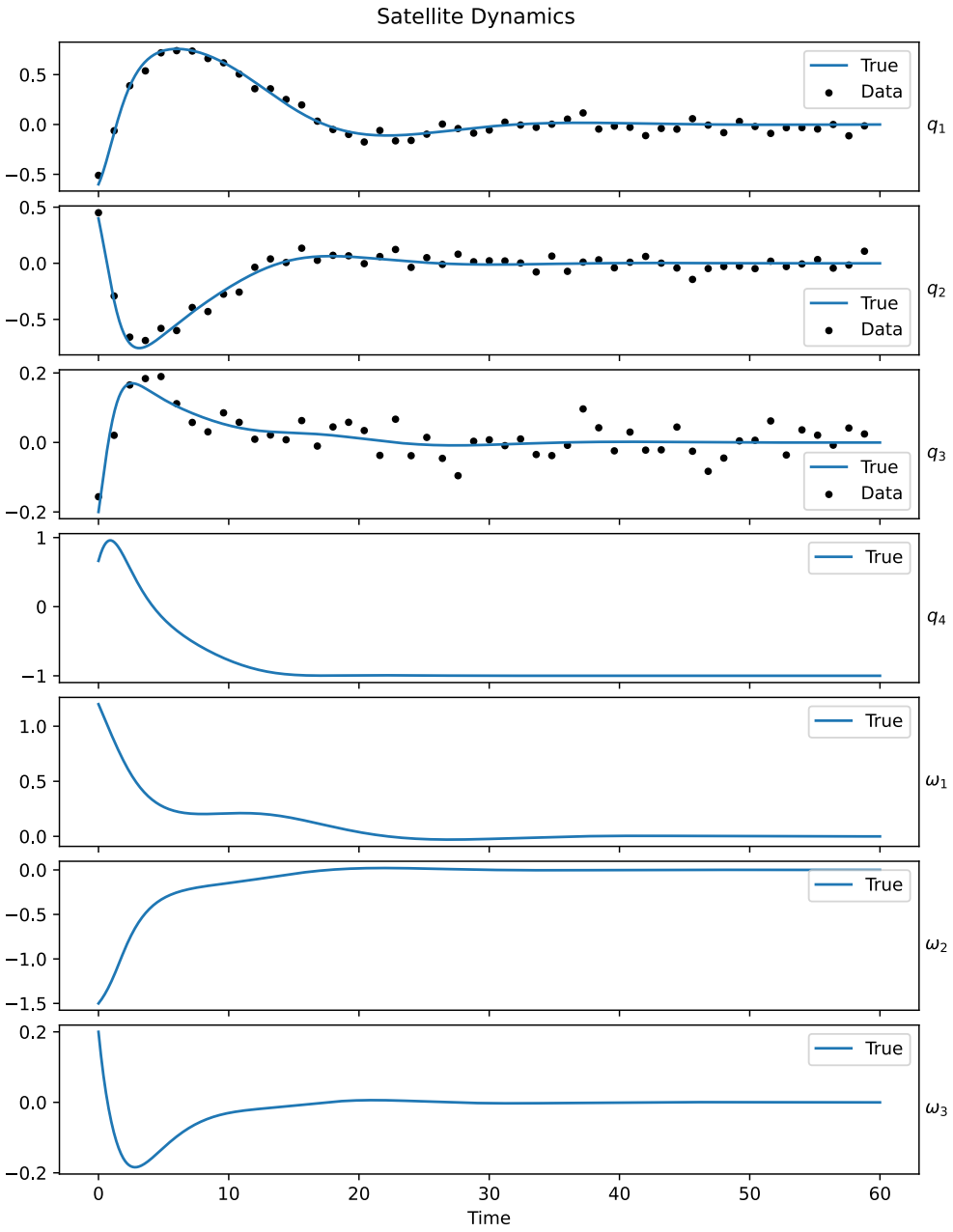

Figure 9: Satellite Dynamics

Problem 1: Parameters are the control coefficients (k1, k2)

1 What is the likelihood model? Please describe how you determine this.

For this exercise, the likelihood model will tell us how likely/probable the observed sensor data is given the parameters of the model (k_1, k_2). We only have access to a subset of all data (q_1, q_2, q_3) that the satellite generates, so we will focus on just the data that is available. Given a function $f(\theta, t)$ that propagates the trajectory of the satellite using the parameters $\theta = (k_1, k_2)$ and outputs the quaternions $(q_1^{(t)}, q_2^{(t)}, q_3^{(t)})$ at any given time t,

Let

$$\text{params: } \theta = (k_1, k_2)$$
$$\text{time instances: } t = 1, 2, ..., 50$$
$$\text{trajectory function: } f(\theta, t) = \begin{bmatrix} q_1^{(t)} \\ q_2^{(t)} \\ q_3^{(t)} \end{bmatrix}$$

We have already observed the sensor data $y^{(t)}$ for the true satellite trajectory at each time instance t.

$$\text{actual sensor data at time t: } y^{(t)} = \begin{bmatrix} y_1^{(t)} \\ y_2^{(t)} \\ y_3^{(t)} \end{bmatrix}$$

Hence, the likelihood can be written as:

$$\text{Likelihood: } \mathcal{L}(\theta) = \mathbb{P}(y|\theta)$$

The sensor captures quaternion data (q_1, q_2, q_3) corrupted by independent Gaussian noise with 0 mean and standard deviation of 0.05. This means the data captured by the sensor of the actual satellite trajectory is not deterministic, but stochastic and follows a probability distribution. The meaning of $\mathbb{P}(y|\theta)$ is that we will be comparing the sensor data $y^{(t)}$ with the trajectory output $q^{(t)}$ by the function $f(\theta, t)$ at each time instance t.

$$\text{sensor noise: } \xi \sim \mathcal{N}(0, 0.05^2)$$
$$\text{distribution of sensor data at time t: } s^{(t)} = \begin{bmatrix} q_1^{(t)} + \xi \\ q_2^{(t)} + \xi \\ q_3^{(t)} + \xi \end{bmatrix}$$

$s^{(t)}$ is a multivariate gaussian distribution with mean $q^{(t)}$ and covariance matrix $0.05^2 I$. This is because the noise is independent and identically distributed for each quaternion component.

$$\text{dimensions: } d = 3$$
$$\text{covariance matrix: } \Sigma = 0.05^2 I = \begin{bmatrix} 0.05^2 & 0 & 0 \\ 0 & 0.05^2 & 0 \\ 0 & 0 & 0.05^2 \end{bmatrix}$$
$$s^{(t)} = \mathcal{N}(q^{(t)}, \Sigma)$$

Now we know that the likelihood for a certain sensor measurement of the quaternions is determined by how far away it is from the quaternions of the trajectory propagated by certain θ, taking into account the noise of the sensor. Different parameters will result in different trajectories, and the sensor data might be closer to some trajectories than others. The likelihood then is the probability of the observed sensor data given a trajectory mapped by the chosen parameters. In other words, we are trying to see how probable/likely the data captured by the sensor is given its noise, if a certain trajectory was mapped by the chosen parameters. The trajectory determined by the parameters is in continuous time, while the sensor data is in discrete time. We need to select the quaternions of the

trajectory at the same time instances as the sensor data. The probability of $y^{(t)}$ given θ is then the probability of a multivariate gaussian distribution with mean $q^{(t)}$ (output of the function $f(\theta, t)$) and covariance matrix $0.05^2 I$ at $y^{(t)}$. The likelihood is then the product of the probabilities of the $y^{(t)}$ given $s^{(t)}$ at each time instance t.

$$\text{Likelihood: } \mathcal{L}(\theta) = \prod_{t=1}^{50} \mathbb{P}(y^{(t)}|s^{(t)})$$

For ease of computation, we will actually use the log-likelihood.

$$\text{Log-Likelihood: } \log \mathcal{L}(\theta) = \sum_{t=1}^{50} \log \mathbb{P}(y^{(t)}|s^{(t)})$$

Since the noise is a multivariate gaussian, we can easily use log probability of a multivariate gaussian to compute this value.

Let

$$\text{determinant of covariance matrix: } |\Sigma|$$

Then

$$\begin{aligned}
\log \mathbb{P}(y^{(t)}|s^{(t)}) &= \log \mathcal{N}(y^{(t)}|q^{(t)}, \Sigma) \\
&= \log \left(\frac{\exp\left(-\frac{1}{2}(y^{(t)} - q^{(t)})^T \Sigma^{-1} (y^{(t)} - q^{(t)})\right)}{\sqrt{(2\pi)^d |\Sigma|}} \right) \\
&= -\frac{d}{2}\log(2\pi) - \frac{1}{2}\log(|\Sigma|) - \frac{1}{2}(y^{(t)} - q^{(t)})^T \Sigma^{-1} (y^{(t)} - q^{(t)}) \\
&= -\frac{1}{2}\left(d\log(2\pi) + \log(|\Sigma|) + (y^{(t)} - q^{(t)})^T \Sigma^{-1} (y^{(t)} - q^{(t)}) \right)
\end{aligned}$$

Note: If the matrices y and q contain all the quaternions for all time instances,

$$y = \begin{bmatrix} y_1^{(1)} & y_1^{(2)} & \dots & y_1^{(50)} \\ y_2^{(1)} & y_2^{(2)} & \dots & y_2^{(50)} \\ y_3^{(1)} & y_3^{(2)} & \dots & y_3^{(50)} \end{bmatrix}$$

$$q = \begin{bmatrix} q_1^{(1)} & q_1^{(2)} & \dots & q_1^{(50)} \\ q_2^{(1)} & q_2^{(2)} & \dots & q_2^{(50)} \\ q_3^{(1)} & q_3^{(2)} & \dots & q_3^{(50)} \end{bmatrix}$$

And the difference $y - q$ is computed element-wise, the log likelihood can be computed in a vectorized form which is explored in the `multigauss_logpdf` function in the code giving a ~1000x speedup over `scipy.stats.multivariate_normal.logpdf` and 10x over `lognormpdf`.

2 What is the form of the posterior?

In Bayesian inference, the posterior is the probability of the parameters(θ) given the observed data(y). It is proportional to the likelihood of the data given the parameters and the prior probability of the parameters.

$$\mathbb{P}(\theta|y) = \frac{\mathbb{P}(\theta)\mathbb{P}(y|\theta)}{\mathbb{P}(y)}$$

Where

$$\begin{aligned}
\mathbb{P}(\theta) &= \text{Prior probability of the parameters} \\
\mathbb{P}(y|\theta) &= \text{Likelihood of the data given the parameters} \\
\mathbb{P}(y) &= \text{Marginal likelihood of the data} \\
\mathbb{P}(\theta|y) &= \text{Posterior probability of the parameters given the data}
\end{aligned}$$

The posterior is basically the prediction of our parameters $\theta = (k_1, k_2)$ using our prior belief about k_1, k_2 (Prior), how probable the data y is given a trajectory propagated by the parameters k_1, k_2 (Likelihood) and the probability of the data y itself(Marginal Likelihood). The Prior for each θ_i is given in the problem statement to be $\theta_i \sim \mathcal{N}(0, 1)$, so we can both sample from it and determine the probability density. The Likelihood can be computed as described in the previous section. The Marginal Likelihood is the probability of the data y given the model of the satellite, which is the integral of their probability over all possible parameters k_1, k_2. Since the space of parameters is unbounded unsolvable analytically, this integral is intractable. This is why we use MCMC methods to sample from the posterior distribution directly, because it is able to sample from a distribution that cannot be be evaluated up to their normalizing constant, which is the marginal likelihood of the data in this case.

In this way, we can compute the posterior and log posterior distribution by:

$$\begin{aligned}
\text{Posterior: } \mathbb{P}(\theta|y) &: \mathbb{P}(\theta)\mathbb{P}(y|\theta) \\
\text{Log Posterior: } \log \mathbb{P}(\theta|y) &: \log \mathbb{P}(\theta) + \log \mathbb{P}(y|\theta)
\end{aligned}$$

Here both the probabilities are multivariate gaussian and can be calculated in the same way as described in the previous section.

Since this problem has a 2-dimensional parameter space, we can visualize the posterior distribution as a 2D plot

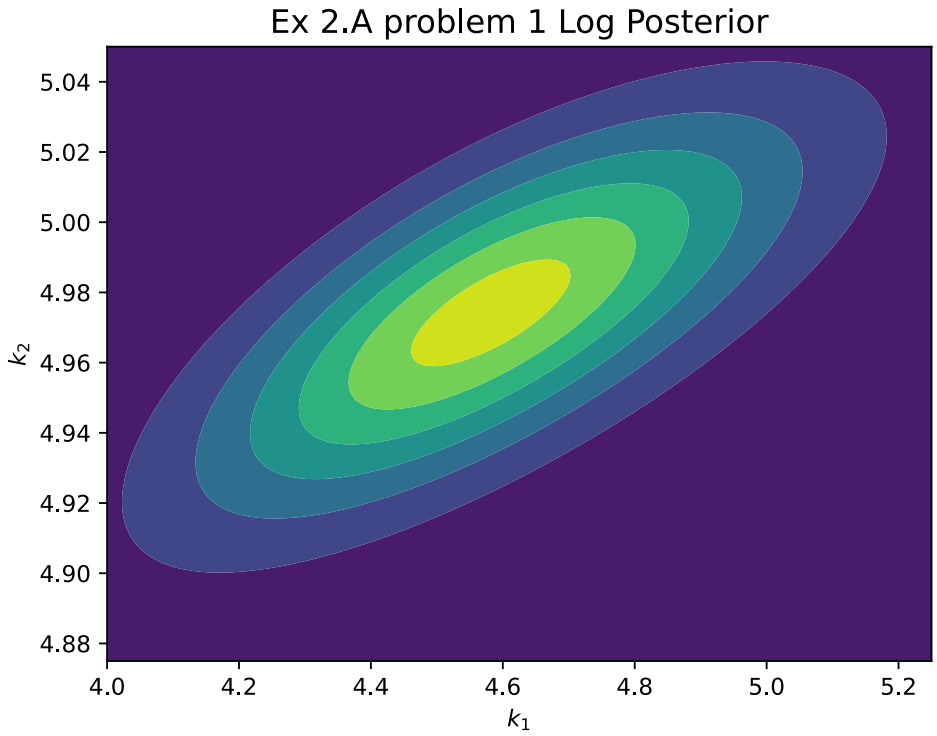

Figure 10: Log Posterior for Problem 1

3 How did you tune your proposal? Think carefully about what a good initial point and initial covariance could be?

The proposal used for the DRAM is a gaussian random walk proposal. For the initial point we can start off from the Maximum Aposteriori(MAP) estimate of the posterior distribution. This will be a certain value of $\theta = (k_1, k_2)$. The MAP estimate can be found by maximizing the logPDF of the posterior distribution. If we simplify the problem and approximate the posterior as a 2D gaussian, the scaled negative inverse of the hessian at the MAP can be used as the covariance matrix for a random walk proposal. This is also called the Laplace Approximation. The scaling factor of the negative inverse hessian is important for determining the initial covariance of the the random walk proposal and has been taken from the suggestion in the notes (ref: eq 16.6). Specifically:

Let

$$\text{Log Posterior: } \log \mathbb{P}(\theta|y) = f_{Xpost}$$

Then

$$\begin{aligned} \theta_{MAP} &= \arg\max_x f_{Xpost} \\ \theta^{(0)} &= \theta_{MAP} \\ \Sigma^{(0)}_{proposal} &= -\frac{2.4^2}{d}\left(\nabla^2 f_{Xpost}(\theta_{MAP})\right)^{-1} \end{aligned}$$

Using `scipy.optimize.minimize` with $-f_{Xpost}$ as the objective function, we find the MAP and initial proposal covariance matrix:

- $\theta^{(0)} = (k_1, k_2)$: (4.58036977, 4.9741691)
- $$\Sigma^{(0)}_{proposal} : \begin{bmatrix} 7.94e-4 & 9.99e-5 \\ 9.99e-5 & 8.45e-4 \end{bmatrix}$$

After the initial $\theta^{(0)} = (k_1^{(0)}, k_2^{(0)})$ and $\Sigma^{(0)}_{proposal}$ are determined, the proposal is tuned by adjusting γ and s_d in the DRAM algorithm as described in part 1 of this project. The covariance of the accepted samples is updated at each iteration and scaled by s_d to get the covariance matrix of the first proposal, and if the first proposal is rejected, the covariance matrix is scaled further by γ for the second proposal.

Since the ideal acceptance ratio is 20-30%, I try to change s_d and γ to get an acceptance ratio in this range. While doing this, I also monitor the Marginals, Autocorrelation vs lag plot, the Integrated Autocorrelation(IAC) and the visual inspection of the samples to ensure that the samples are mixing well and the proposal is tuned correctly. Specifically

- Acceptance ratio:
  - if low, decrease s_d and γ
  - if high, increase s_d and γ
- Marginals:
  - if clustering around a few points and not exploring enough, increase s_d and γ
- Autocorrelation:
  - if doesnt decay quickly, increase s_d and γ
- IAC:
  - if high, increase s_d and γ

For the sake of speed, to increase the acceptance ratio I preferred to decrease s_d before I increased γ, because if the first proposal is rejected, the trajectory has to be propagated again for the second proposal, which is computationally expensive. Hence I prefered the first proposal to be accepted more often rather than it rejecting and the 2nd proposal being accepted. In fact, I experimented with increasing γ as high as 100, but it has minimal impact on the acceptance ratio, so I kept it low and preferred to tune s_d instead. This makes sense given that increasing s_d will also increase the scale for the second proposal as a side effect, thus affecting both proposals and having a larger impact. γ is meant to be used when the first proposal rejected for being too exploratory, so it should ideally be ≤ 1 so the second proposal is more conservative.

After tuning, I ended up with $s_d = 9$ and $\gamma = 0.9$ for the final proposal. I started off with a high acceptance ratio, so I increased s_d to get the acceptance ratio in the desired range. I also increased γ_{DRAM} slightly. I made sure that all other metrics remained good while this happened.

4/5 Analyze your results using the same deliverables as you used in Section 1.

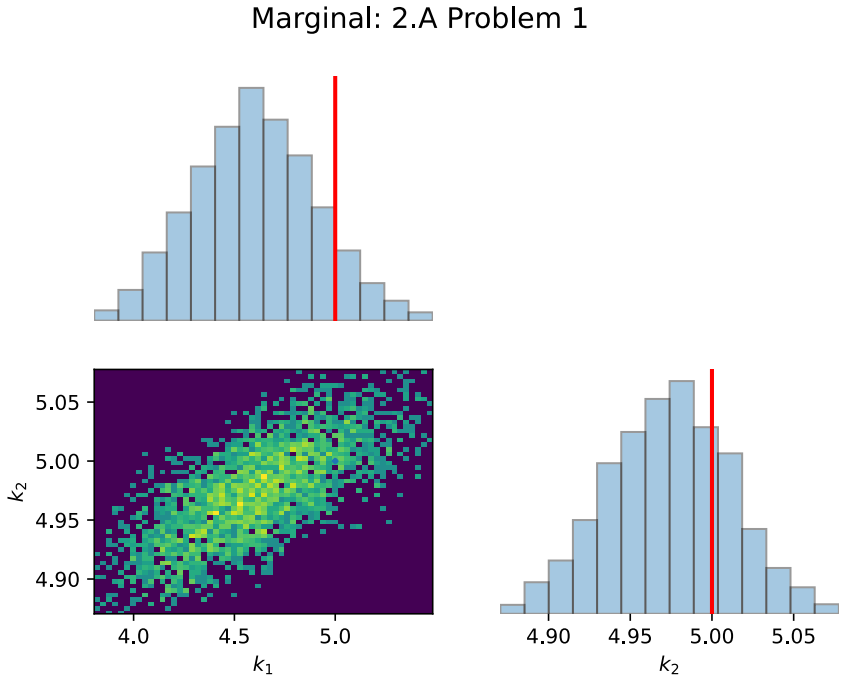

Figure 11: Marginals for Problem 1

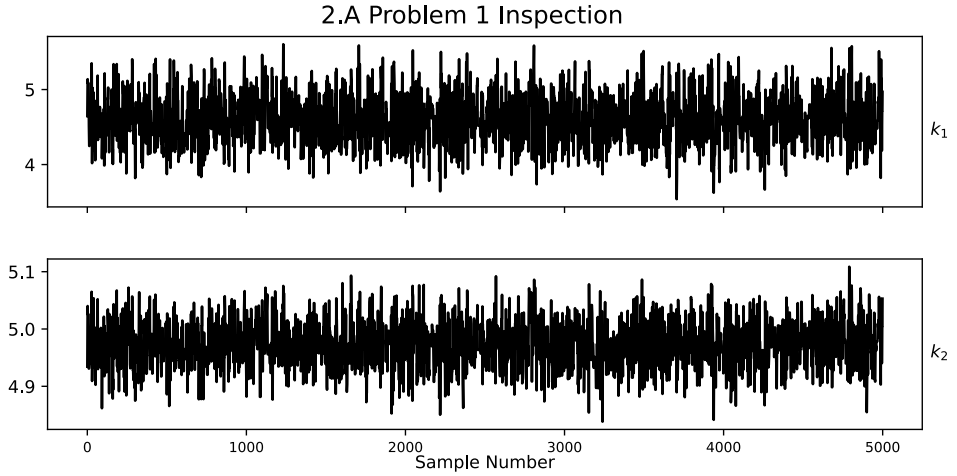

Figure 12: Visual Inspection for Problem 1

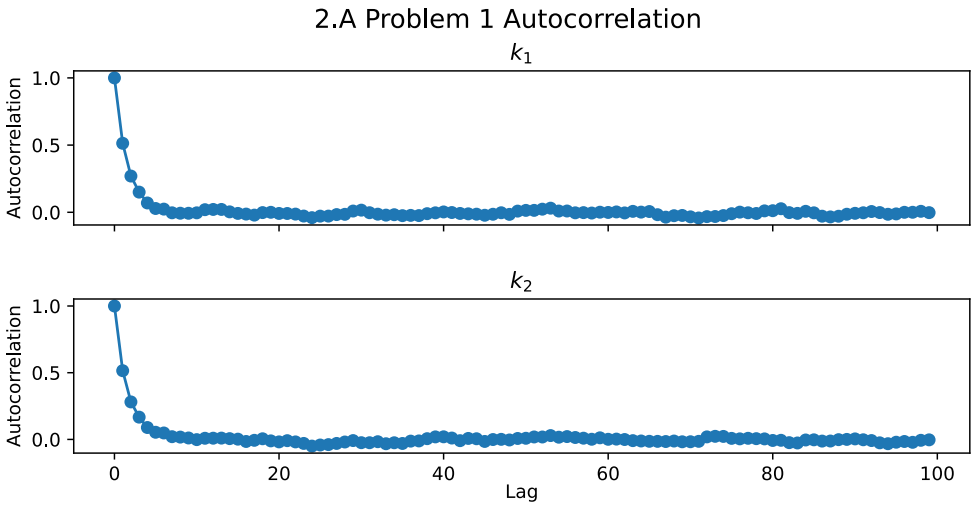

Figure 13: Autocorrelation for Problem 1

- Integrated autocorrelation values:
  - k_1: 1.77
  - k_2: 2.4
- Acceptance ratio: 31%

Right off the bat, we can see that the 2-D scatter plot of the posterior matches the plot made in section 2 where we plotted the posterior, so our MCMC has worked well. The acceptance ratio started off very high, but was tuned to 30% while making sure that the samples are mixing well. We can see that the marginals look good, and the Autocorrelation plot decays quickly leading the low values of IAC. Finally, the visual inspection of the samples shows that the samples do not get stuck in a local area and the MC is exploring the parameter space well.

When we compare the value of the true k_1 and k_2 with the marginals, we can see that the true values are not exactly at the peak of the gaussian marginals. This could be due to there being noise in our data, and the fact that our prior was very far of from the true values. Given these limitation, we cannot expect the posterior to converge exactly to the true values, but we can see that true values are not that far off from the peak of the marginals, which is a good sign.

6 Plot the prior and posterior predictives of the dynamics (separately)

The true trajectory is plotted in blue and the predicted trajectory is plotted in gray.

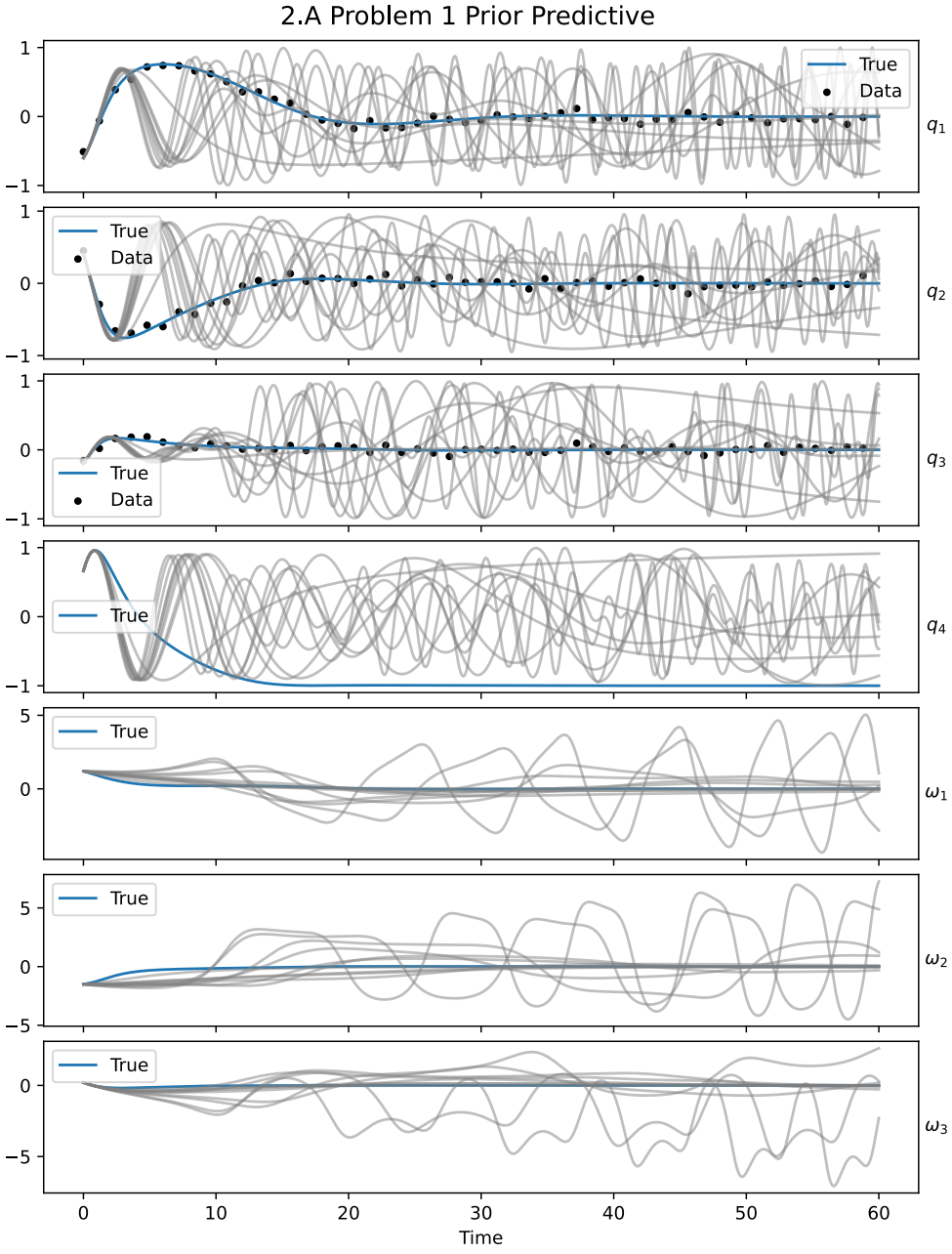

Figure 14: Prior Predictive for Problem 1

While the prior predictive matches the trajectory initially, it soon deviates really far away from the true satellite trajectory. This is expected since we are not using any data to infer our parameters at this stage. The posterior predictive trajectory very closely matches the true satellite trajectory, being almost on top of the true trajectory which is a good sign that our MCMC has worked well and the parameters have been inferred correctly. With these plots, we can clearly see the difference Bayesian inference has made in our predictions.

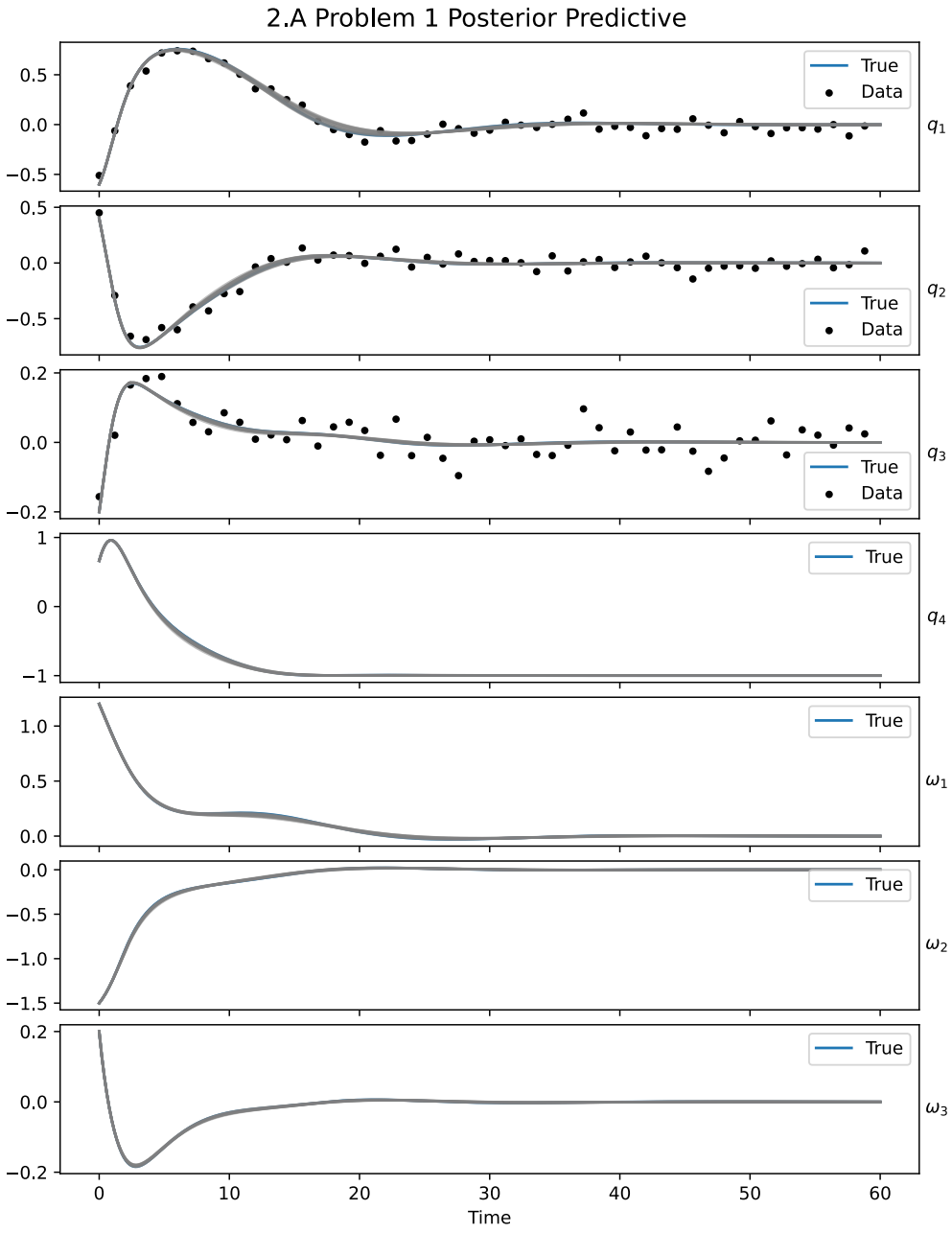

Figure 15: Posterior Predictive for Problem 1

Problem 2: Parameters are the control coefficients and a product of inertia (k1, k2, J12)

1 What is the likelihood model? Please describe how you determine this.

For this exercise, the likelihood model will tell us how likely/probable the observed sensor data is given the parameters of the model (k_1, k_2, J_{12}). We only have access to a subset of all data (q_1, q_2, q_3) that the satellite generates, so we will focus on just the data that is available. Given a function $f(\theta, t)$ that propagates the trajectory of the satellite using the parameters $\theta = (k_1, k_2, J_{12})$ and outputs the quaternions $(q_1^{(t)}, q_2^{(t)}, q_3^{(t)})$ at any given time t,

Let

$$\text{params: } \theta = (k_1, k_2, J_{12})$$
$$\text{time instances: } t = 1, 2, ..., 50$$
$$\text{trajectory function: } f(\theta, t) = \begin{bmatrix} q_1^{(t)} \\ q_2^{(t)} \\ q_3^{(t)} \end{bmatrix}$$

We have already observed the sensor data $y^{(t)}$ for the true satellite trajectory at each time instance t.

$$\text{actual sensor data at time t: } y^{(t)} = \begin{bmatrix} y_1^{(t)} \\ y_2^{(t)} \\ y_3^{(t)} \end{bmatrix}$$

Hence, the likelihood can be written as:

$$\text{Likelihood: } \mathcal{L}(\theta) = \mathbb{P}(y|\theta)$$

The sensor captures quaternion data (q_1, q_2, q_3) corrupted by independent Gaussian noise with 0 mean and standard deviation of 0.05. This means the data captured by the sensor of the actual satellite trajectory is not deterministic, but stochastic and follows a probability distribution. The meaning of $\mathbb{P}(y|\theta)$ is that we will be comparing the sensor data $y^{(t)}$ with the trajectory output $q^{(t)}$ by the function $f(\theta, t)$ at each time instance t.

$$\text{sensor noise: } \xi \sim \mathcal{N}(0, 0.05^2)$$
$$\text{distribution of sensor data at time t: } s^{(t)} = \begin{bmatrix} q_1^{(t)} + \xi \\ q_2^{(t)} + \xi \\ q_3^{(t)} + \xi \end{bmatrix}$$

$s^{(t)}$ is a multivariate gaussian distribution with mean $q^{(t)}$ and covariance matrix $0.05^2 I$. This is because the noise is independent and identically distributed for each quaternion component.

$$\text{dimensions: } d = 3$$
$$\text{covariance matrix: } \Sigma = 0.05^2 I = \begin{bmatrix} 0.05^2 & 0 & 0 \\ 0 & 0.05^2 & 0 \\ 0 & 0 & 0.05^2 \end{bmatrix}$$
$$s^{(t)} = \mathcal{N}(q^{(t)}, \Sigma)$$

Now we know that the likelihood for a certain sensor measurement of the quaternions is determined by how far away it is from the quaternions of the trajectory propagated by certain θ, taking into account the noise of the sensor. Different parameters will result in different trajectories, and the sensor data might be closer to some trajectories than others. The likelihood then is the probability of the observed sensor data given a trajectory mapped by the chosen parameters. In other words, we are trying to see how probable/likely the data captured by the sensor is given its noise, if a certain trajectory was mapped by the chosen parameters. The trajectory determined by the

parameters is in continuous time, while the sensor data is in discrete time. We need to select the quaternions of the trajectory at the same time instances as the sensor data. The probability of $y^{(t)}$ given θ is then the probability of a multivariate gaussian distribution with mean $q^{(t)}$ (output of the function $f(\theta, t)$) and covariance matrix $0.05^2 I$ at $y^{(t)}$. The likelihood is then the product of the probabilities of the $y^{(t)}$ given $s^{(t)}$ at each time instance t.

$$\text{Likelihood: } \mathcal{L}(\theta) = \prod_{t=1}^{50} \mathbb{P}(y^{(t)}|s^{(t)})$$

For ease of computation, we will actually use the log-likelihood.

$$\text{Log-Likelihood: } \log \mathcal{L}(\theta) = \sum_{t=1}^{50} \log \mathbb{P}(y^{(t)}|s^{(t)})$$

Since the noise is a multivariate gaussian, we can easily use log probability of a multivariate gaussian to compute this value.

Let

$$\text{determinant of covariance matrix: } |\Sigma|$$

Then

$$\begin{aligned}
\log \mathbb{P}(y^{(t)}|s^{(t)}) &= \log \mathcal{N}(y^{(t)}|q^{(t)}, \Sigma) \\
&= \log \left(\frac{\exp\left(-\frac{1}{2}(y^{(t)} - q^{(t)})^T \Sigma^{-1} (y^{(t)} - q^{(t)})\right)}{\sqrt{(2\pi)^d |\Sigma|}} \right) \\
&= -\frac{d}{2}\log(2\pi) - \frac{1}{2}\log(|\Sigma|) - \frac{1}{2}(y^{(t)} - q^{(t)})^T \Sigma^{-1} (y^{(t)} - q^{(t)}) \\
&= -\frac{1}{2}\left(d\log(2\pi) + \log(|\Sigma|) + (y^{(t)} - q^{(t)})^T \Sigma^{-1} (y^{(t)} - q^{(t)}) \right)
\end{aligned}$$

Note: If the matrices y and q contain all the quaternions for all time instances,

$$y = \begin{bmatrix} y_1^{(1)} & y_1^{(2)} & \dots & y_1^{(50)} \\ y_2^{(1)} & y_2^{(2)} & \dots & y_2^{(50)} \\ y_3^{(1)} & y_3^{(2)} & \dots & y_3^{(50)} \end{bmatrix}$$

$$q = \begin{bmatrix} q_1^{(1)} & q_1^{(2)} & \dots & q_1^{(50)} \\ q_2^{(1)} & q_2^{(2)} & \dots & q_2^{(50)} \\ q_3^{(1)} & q_3^{(2)} & \dots & q_3^{(50)} \end{bmatrix}$$

And the difference $y - q$ is computed element-wise, the log likelihood can be computed in a vectorized form which is explored in the `multigauss_logpdf` function in the code giving a ~1000x speedup over `scipy.stats.multivariate_normal.logpdf` and 10x over `lognormpdf`.

2 What is the form of the posterior?

In Bayesian inference, the posterior is the probability of the parameters(θ) given the observed data(y). It is proportional to the likelihood of the data given the parameters and the prior probability of the parameters.

$$\mathbb{P}(\theta|y) = \frac{\mathbb{P}(\theta)\mathbb{P}(y|\theta)}{\mathbb{P}(y)}$$

Where

$$\begin{aligned}
\mathbb{P}(\theta) &= \text{Prior probability of the parameters} \\
\mathbb{P}(y|\theta) &= \text{Likelihood of the data given the parameters} \\
\mathbb{P}(y) &= \text{Marginal likelihood of the data} \\
\mathbb{P}(\theta|y) &= \text{Posterior probability of the parameters given the data}
\end{aligned}$$

The posterior is basically the prediction of our parameters $\theta = (k_1, k_2, J_{12})$ using our prior belief about k_1, k_2, J_{12} (Prior), how probable the data y is given a trajectory propagated by the parameters k_1, k_2, J_{12} (Likelihood) and the probability of the data y itself(Marginal Likelihood). The Prior for each θ_i is given in the problem statement to be $\theta_i \sim \mathcal{N}(0, 1)$, so we can both sample from it and determine the probability density. The Likelihood can be computed as described in the previous section. The Marginal Likelihood is the probability of the data y given the model of the satellite, which is the integral of their probability over all possible parameters k_1, k_2, J_{12}. Since the space of parameters is unbounded unsolvable analytically, this integral is intractable. This is why we use MCMC methods to sample from the posterior distribution directly, because it is able to sample from a distribution that cannot be be evaluated up to their normalizing constant, which is the marginal likelihood of the data in this case.

In this way, we can compute the posterior and log posterior distribution by:

$$\begin{aligned}
\text{Posterior: } \mathbb{P}(\theta|y) &: \mathbb{P}(\theta)\mathbb{P}(y|\theta) \\
\text{Log Posterior: } \log \mathbb{P}(\theta|y) &: \log \mathbb{P}(\theta) + \log \mathbb{P}(y|\theta)
\end{aligned}$$

Here both the probabilities are multivariate gaussian and can be calculated in the same way as described in the previous section.

3 How did you tune your proposal? Think carefully about what a good initial point and initial covariance could be?

The proposal used for the DRAM is a gaussian random walk proposal. For the initial point we can start off from the Maximum Aposteriori(MAP) estimate of the posterior distribution. This will be a certain value of $\theta = (k_1, k_2, J_{12})$. The MAP estimate can be found by maximizing the logPDF of the posterior distribution. If we simplify the problem and approximate the posterior as a 3D gaussian, the scaled negative inverse of the hessian at the MAP can be used as the covariance matrix for a random walk proposal. This is also called the Laplace Approximation. The scaling factor of the negative inverse hessian is important for determining the initial covariance of the the random walk proposal and has been taken from the suggestion in the notes (ref: eq 16.6). Specifically:

Let

$$\text{Log Posterior: } \log \mathbb{P}(\theta|y) = f_{Xpost}$$

Then

$$\begin{aligned}
\theta_{MAP} &= \arg\max_x f_{Xpost} \\
\theta^{(0)} &= \theta_{MAP} \\
\Sigma^{(0)}_{proposal} &= -\frac{2.4^2}{d}\left(\nabla^2 f_{Xpost}(\theta_{MAP})\right)^{-1}
\end{aligned}$$

Using `scipy.optimize.minimize` with $-f_{Xpost}$ as the objective function, we find the MAP and initial proposal covariance matrix:

- $\theta^{(0)} = (k_1, k_2, J_{12})$: (4.55097817, 4.87289835, 1.59590382)
- $$\Sigma^{(0)}_{proposal} : \begin{bmatrix} 0.416 & 0.047 & -0.041 \\ 0.047 & 0.012 & -0.028 \\ 0.041 & -0.028 & 0.097 \end{bmatrix}$$

After the initial $\theta^{(0)} = (k_1^{(0)}, k_2^{(0)}, J_{12})$ and $\Sigma_{proposal}^{(0)}$ are determined, the proposal is tuned by adjusting γ and s_d in the DRAM algorithm as described in part 1 of this project. The covariance of the accepted samples is updated at each iteration and scaled by s_d to get the covariance matrix of the first proposal, and if the first proposal is rejected, the covariance matrix is scaled further by γ for the second proposal.

Since the ideal acceptance ratio is 20-30%, I try to change s_d and γ to get an acceptance ratio in this range. While doing this, I also monitor the Marginals, Autocorrelation vs lag plot, the Integrated Autocorrelation(IAC) and the visual inspection of the samples to ensure that the samples are mixing well and the proposal is tuned correctly. Specifically

- Acceptance ratio:
  - if low, decrease s_d and γ
  - if high, increase s_d and γ
- Marginals:
  - if clustering around a few points and not exploring enough, increase s_d and γ
- Autocorrelation:
  - if doesnt decay quickly, increase s_d and γ
- IAC:
  - if high, increase s_d and γ

For the sake of speed, to increase the acceptance ratio I preferred to decrease s_d before I increased γ, because if the first proposal is rejected, the trajectory has to be propagated again for the second proposal, which is computationally expensive. Hence I prefered the first proposal to be accepted more often rather than it rejecting and the 2nd proposal being accepted. In fact, I experimented with increasing γ as high as 100, but it has minimal impact on the acceptance ratio, so I kept it low and preferred to tune s_d instead. This makes sense given that increasing s_d will also increase the scale for the second proposal as a side effect, thus affecting both proposals and having a larger impact. γ is meant to be used when the first proposal rejected for being too exploratory, so it should ideally be ≤ 1 so the second proposal is more conservative.

After tuning, I ended up with $s_d = 4$ and $\gamma = 0.9$ for the final proposal. I started off with a high acceptance ratio, so I increased s_d to get the acceptance ratio in the desired range. I also increased γ_{DRAM} slightly. I made sure that all other metrics remained good while this happened.

4/5 Analyze your results using the same deliverables as you used in Section 1.

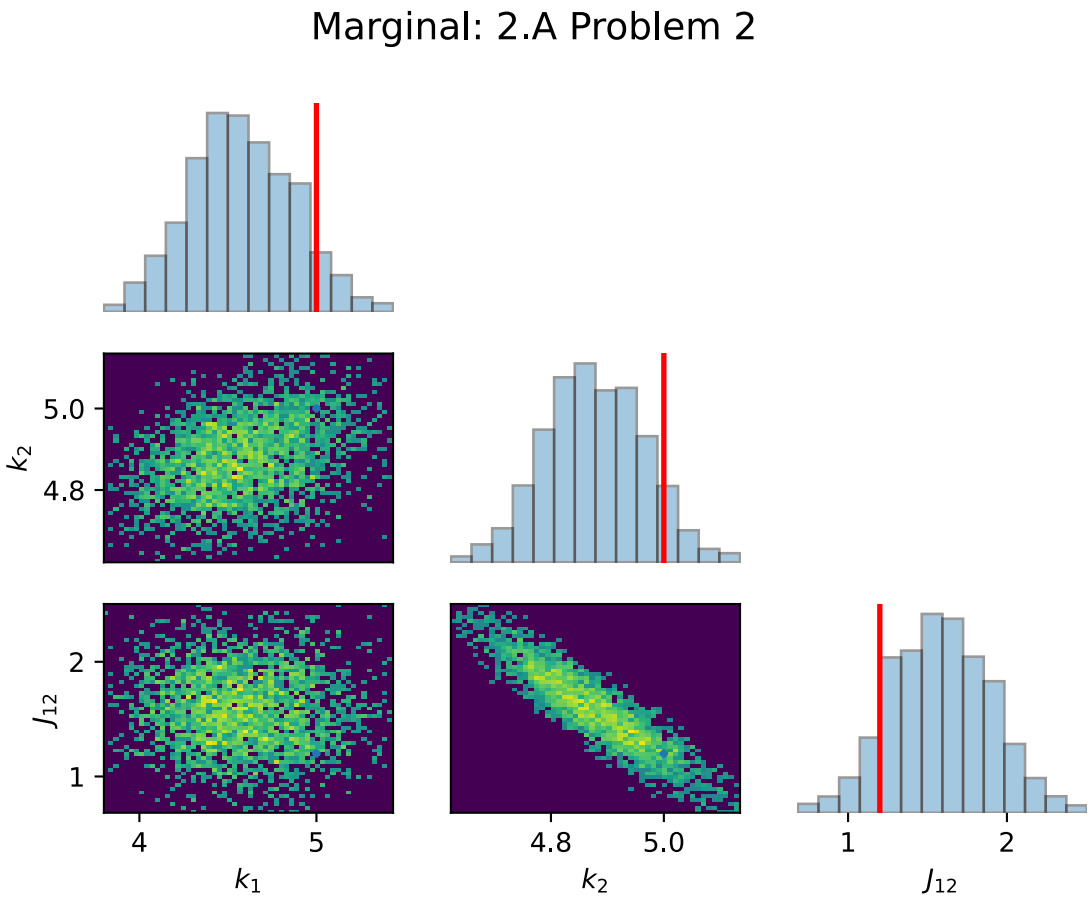

Figure 16: Marginals for Problem 2

Integrated autocorrelation values:

- Integrated autocorrelation values:
  - k_1: 2.07
  - k_2: 5.97
  - J_{12}: 5.99
- Acceptance ratio: 30%

The acceptance ratio started off quite high, and it was tuned down close to 30% and we can see that the samples are still mixing well. We can see that the marginals look good and smooth, describing gaussian distributions, and the Autocorrelation plot decays quickly leading the low values of IAC. Finally, the visual inspection of the samples shows that the samples do not get stuck in a local area and the MC is exploring the parameter space well.

When we compare the value of the true k_1, k_2 and J_{12} with the marginals, we can see that the true values are not exactly at the peak of the gaussian marginals. This could be due to there being noise in our data, which is significant for J_{12}, and the fact that our prior for k_1 and k_2 was very far of from the true values. Given these limitation, we cannot expect the posterior to converge exactly to the true values, but we can see that true values are not that far off from the peak of the marginals, which is a good sign.

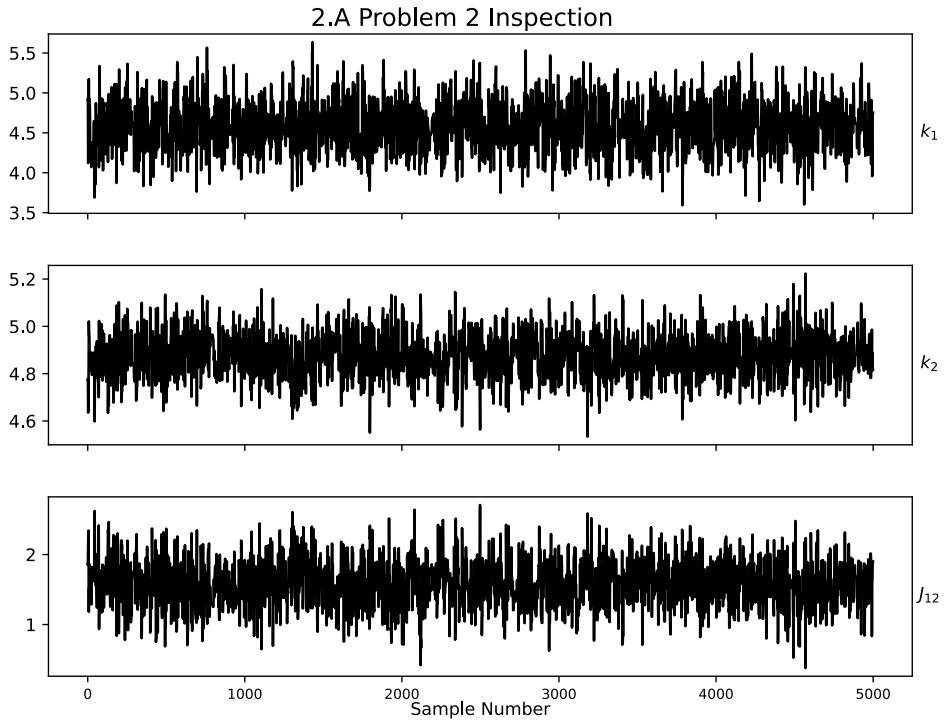

Figure 17: Visual Inspection for Problem 2

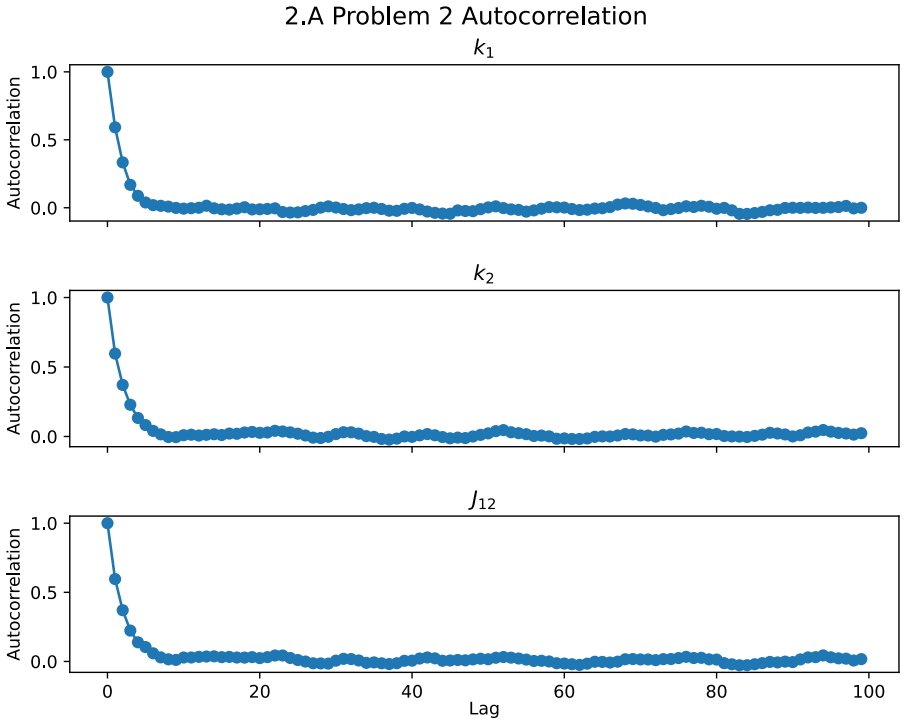

Figure 18: Autocorrelation for Problem 2

6 Plot the prior and posterior predictives of the dynamics (separately)

The true data is in blue and the predicted data is in grey

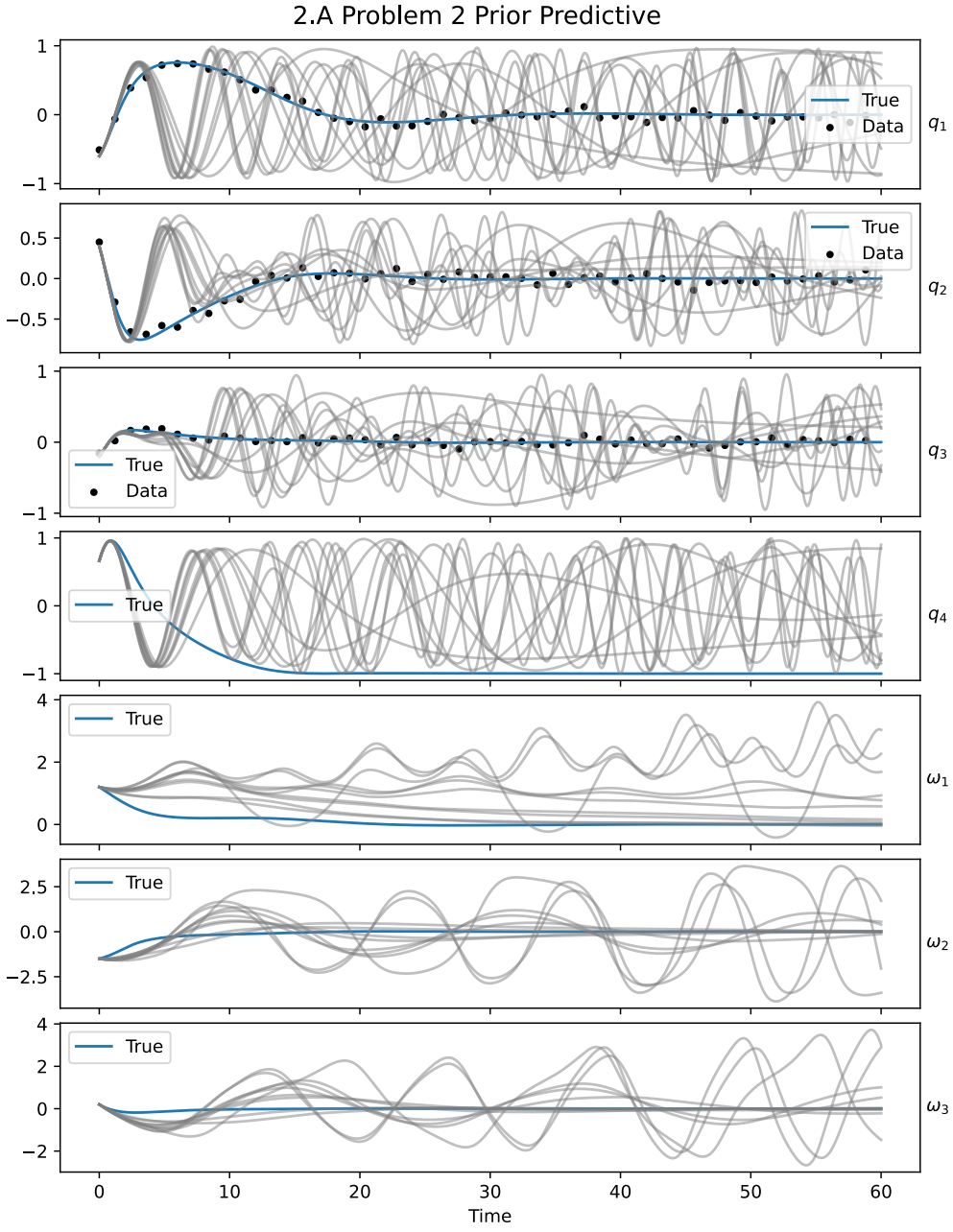

Figure 19: Prior Predictive for Problem 2

The prior predictive looks totally random, only being close to the true trajectory in the beginning while the posterior predictive actually matches the true trajectory of the satellite quite closely. This is a good sign that our model is working well and the parameters are being inferred correctly. The difference between the prior and posterior predictives is quite stark, showing the power of Bayesian inference given that after learning the parameters we can make much better predictions.

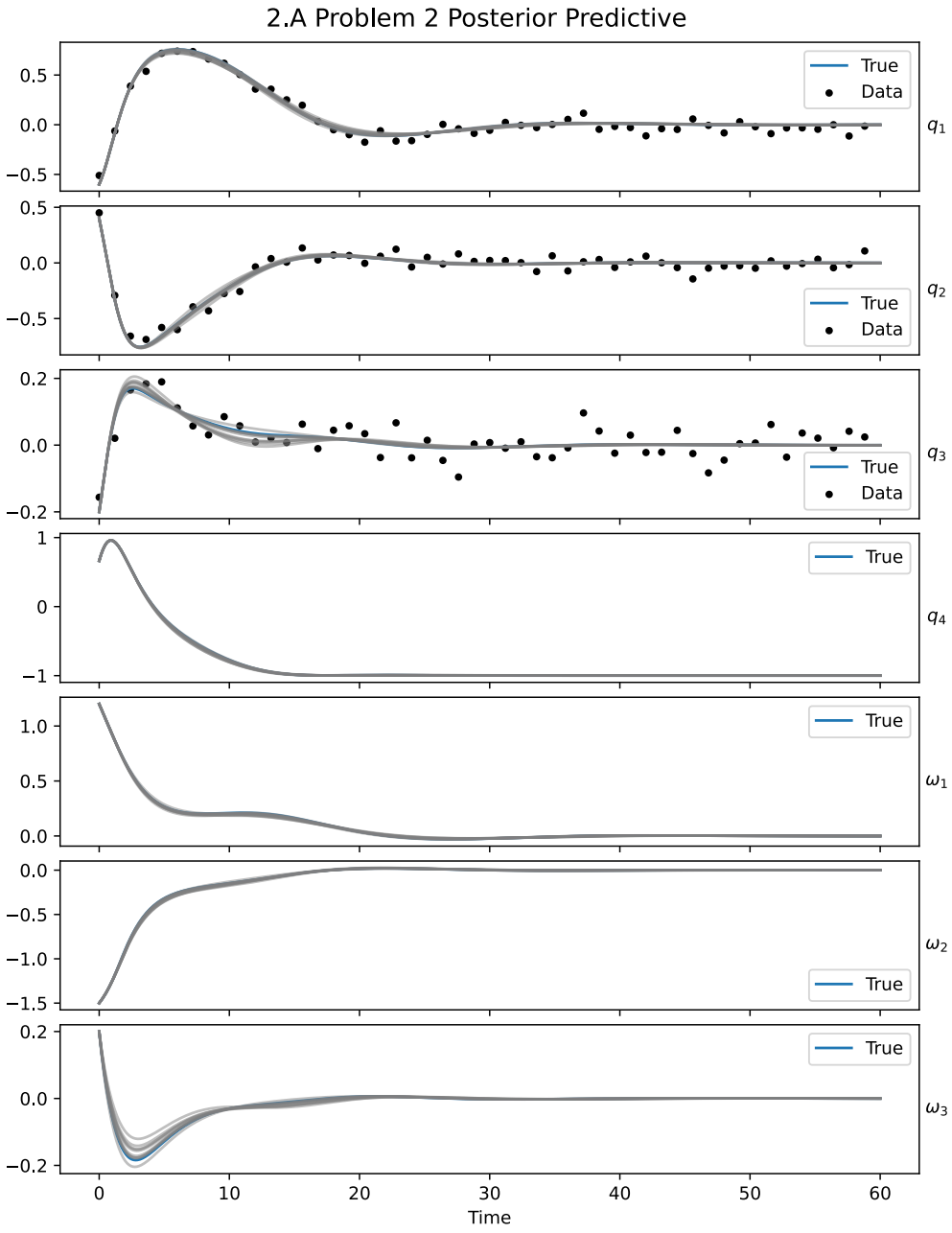

Figure 20: Posterior Predictive for Problem 2

Please comment on the following:

- What is the difference between the two parameter inference problems?

The difference is that in the first problem, we were just trying to infer the control parameters, while in the second problem, the dynamics of the satellite that are described by the product of inertia is also unknown. This makes it a more difficult problem, since the dynamics of the satellite are also unknown and have to be inferred. If the satellite was inherently stable, with known dynamics it might still end up near the true trajectory, but in problem 2, the dynamics are also unknown and can lead to huge deviations from the true trajectory.

- How does the posterior predictive change? - Are there any notable differences?

The posterior predictive changes slightly between the 2 problems, where in problem 1 it very closely matches the true trajectory during the whole time period, while in problem 2, it is a bit off in the beginning but then closely matches the true trajectory, especially for q_3 and ω_3. However, both the predictives are much better than the prior predictive, which is totally random and does not match the true trajectory at all.

2.B SIR Model

After setting up the SIR Model, we can see the observed evolution of the disease and the data points in the plot below. This is done using the true params of the identifiable model since we cannot generate the true evolution using the non-identifiable model.

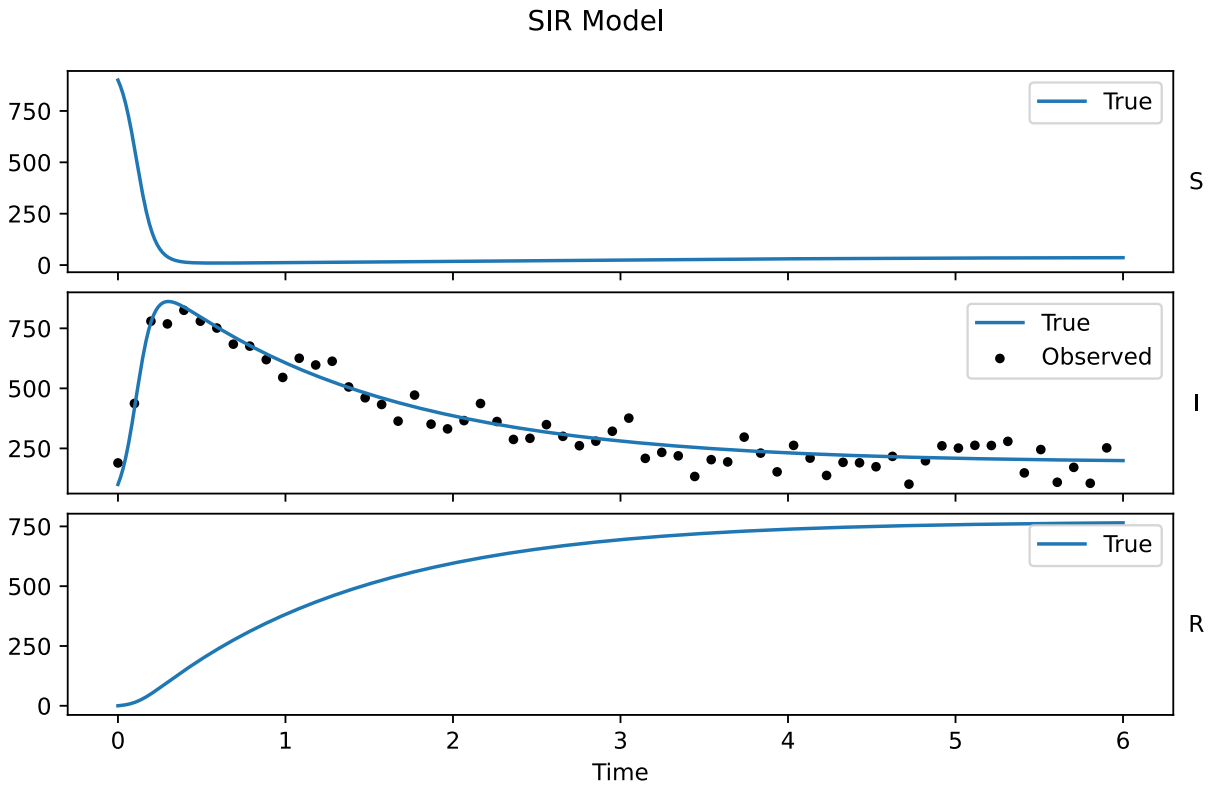

Figure 21: SIR Model Evolution

Problem 1 (Identifiable): Parameters are $\theta = (\beta, r, \delta)$

1 What is the likelihood model? Please describe how you determine this.

For this exercise, the likelihood model will tell us how likely/probable the observed data is given the parameters of the model (β, r, δ). We only have access to a subset of all data (only I from S, I and R) that the evolution trajectory generates, so we will focus on just the data that is available. Given a function $f(\theta, t)$ that evolves the trajectory of the disease model using the parameters $\theta = (\beta, r, \delta)$ and outputs the Infected $(I^{(t)})$ at any given time t,

Let

$$\begin{aligned} \text{params: } & \theta = (\beta, r, \delta) \\ \text{time instances: } & t = 1, 2, ..., 61 \\ \text{trajectory function: } & f(\theta, t) = I^{(t)} \end{aligned}$$

We have already observed the data $y^{(t)}$ for the true trajectory at each time instance t.

$$\text{actual observed data at time t: } y^{(t)}$$

Hence, the likelihood can be written as:

$$\text{Likelihood: } \mathcal{L}(\theta) = \mathbb{P}(y|\theta)$$

We observe data (I) corrupted by independent Gaussian noise with 0 mean and standard deviation of 50. This means the data observed of the actual disease trajectory is not deterministic, but stochastic and follows a probability distribution. The meaning of $\mathbb{P}(y|\theta)$ is that we will be comparing the observed data $y^{(t)}$ with the trajectory output $I^{(t)}$ by the function $f(\theta, t)$ at each time instance t.

$$\text{data noise: } \xi \sim \mathcal{N}(0, 50^2)$$
$$\text{distribution of observed data at time t: } o^{(t)} = I^{(t)} + \xi$$

$o^{(t)}$ is a gaussian distribution with mean $I^{(t)}$ and variance 50^2.

$$o^{(t)} = \mathcal{N}(I^{(t)}, \Sigma)$$

Now we know that the likelihood for a certain observation of the infected population is determined by how far away it is from the infected population of the trajectory evolved by certain θ, taking into account the noise of in our observations. Different parameters will result in different evolutions, and the observed data might be closer to some evolutions than others. The likelihood then is the probability of the observed data given a trajectory evolved by the chosen parameters. In other words, we are trying to see how probable/likely the data of the infected population is given its noise, if a certain trajectory was evolved by the chosen parameters. The trajectory determined by the parameters is in continuous time, while the observed data is in discrete time. We need to select the infected population of the evolution at the same time instances as the observed data. The probability of $y^{(t)}$ given θ is then the probability of the gaussian distribution with mean $I^{(t)}$ (output of the function $f(\theta, t)$) and variance 50^2 at $y^{(t)}$. The likelihood is then the product of the probabilities of the $y^{(t)}$ given $o^{(t)}$ at each time instance t.

$$\text{Likelihood: } \mathcal{L}(\theta) = \prod_{t=1}^{61} \mathbb{P}(y^{(t)}|o^{(t)})$$

For ease of computation, we will actually use the log-likelihood.

$$\text{Log-Likelihood: } \log \mathcal{L}(\theta) = \sum_{t=1}^{61} \log \mathbb{P}(y^{(t)}|o^{(t)})$$

Since the noise is gaussian, we can easily use log probability of a gaussian to compute this value.

Let

$$\sigma = 50$$

Then

$$\begin{aligned}
\log \mathbb{P}(y^{(t)}|o^{(t)}) &= \log \mathcal{N}(y^{(t)}|I^{(t)}, \Sigma) \\
&= \log \left(\frac{\exp\left(-\frac{1}{2}(\frac{y^{(t)} - I^{(t)}}{\sigma})^2\right)}{\sigma\sqrt{2\pi}} \right) \\
&= -\frac{1}{2}\log(2\pi) - \log(\sigma) - \frac{1}{2}\left(\frac{y^{(t)} - I^{(t)}}{\sigma}\right)^2
\end{aligned}$$

We can then plug this into the log-likelihood function to compute the log-likelihood of the data given the parameters for all time instances.

2 What is the form of the posterior?

In Bayesian inference, the posterior is the probability of the parameters(θ) given the observed data(y). It is proportional to the likelihood of the data given the parameters and the prior probability of the parameters.

$$\mathbb{P}(\theta|y) = \frac{\mathbb{P}(\theta)\mathbb{P}(y|\theta)}{\mathbb{P}(y)}$$

Where

$$\begin{aligned}
\mathbb{P}(\theta) &= \text{Prior probability of the parameters} \\
\mathbb{P}(y|\theta) &= \text{Likelihood of the data given the parameters} \\
\mathbb{P}(y) &= \text{Marginal likelihood of the data} \\
\mathbb{P}(\theta|y) &= \text{Posterior probability of the parameters given the data}
\end{aligned}$$

The posterior is basically the prediction of our parameters $\theta = (\beta, r, \delta)$ using our prior belief about (β, r, δ) (Prior), how probable the data y is given a trajectory propagated by the parameters (β, r, δ) (Likelihood) and the probability of the data y itself(Marginal Likelihood). The Prior for each θ_i is given in the problem statement to be $\theta_i \sim \mathcal{N}(0, 1)$, so we can both sample from it and determine the probability density. The Likelihood can be computed as described in the previous section. The Marginal Likelihood is the probability of the data y given the SIR model, which is the integral of its probability over all possible parameters (β, r, δ). Since the space of parameters is unbounded unsolvable analytically, this integral is intractable. This is why we use MCMC methods to sample from the posterior distribution directly, because it is able to sample from a distribution that cannot be be evaluated up to their normalizing constant, which is the marginal likelihood of the data in this case.

In this way, we can compute the posterior and log posterior distribution by:

$$\begin{aligned}
\text{Posterior: } \mathbb{P}(\theta|y) &: \mathbb{P}(\theta)\mathbb{P}(y|\theta) \\
\text{Log Posterior: } \log \mathbb{P}(\theta|y) &: \log \mathbb{P}(\theta) + \log \mathbb{P}(y|\theta)
\end{aligned}$$

Here the prior is a multivariate gaussian but with an identity covariance, which means we can treat all the parameters are independent and calculate the logPDF for each parameter independently.The likelihood is a gaussian as well, as described in the previous section and can be calculated as described in the previous section.

3 How did you tune your proposal? Think carefully about what a good initial point and initial covariance could be?

The proposal used for the DRAM is a gaussian random walk proposal. For the initial point we can start off from the Maximum Aposteriori(MAP) estimate of the posterior distribution. This will be a certain value of $\theta = (\beta, r, \delta)$. The MAP estimate can be found by maximizing the logPDF of the posterior distribution. If we simplify the problem and approximate the posterior as a 3D gaussian, the scaled negative inverse of the hessian at the MAP can be used as the covariance matrix for a random walk proposal. This is also called the Laplace Approximation. The scaling factor of the negative inverse hessian is important for determining the initial covariance of the the random walk proposal and has been taken from the suggestion in the notes (ref: eq 16.6). Specifically:

Let

$$\text{Log Posterior: } \log \mathbb{P}(\theta|y) = f_{Xpost}$$

Then

$$\begin{aligned}
\theta_{MAP} &= \arg\max_x f_{Xpost} \\
\theta^{(0)} &= \theta_{MAP} \\
\Sigma^{(0)}_{proposal} &= -\frac{2.4^2}{d}\left(\nabla^2 f_{Xpost}(\theta_{MAP})\right)^{-1}
\end{aligned}$$

Using `scipy.optimize.minimize` with $-f_{Xpost}$ as the objective function, we find the MAP and initial proposal covariance matrix:

- $\theta^{(0)} = (\beta, r, \delta)$: (0.02 0.61 0.14)
- $$\Sigma^{(0)}_{proposal} : \begin{bmatrix} 4.82564094e-06 & -6.58406798e-06 & -4.59524017e-06 \\ -6.58406798e-06 & 1.23289295e-03 & 6.73081629e-04 \\ -4.59524017e-06 & 6.73081629e-04 & 5.56126832e-04 \end{bmatrix}$$

After the initial $\theta^{(0)} = (\beta^{(0)}, r^{(0)}, \delta^{(0)})$ and $\Sigma^{(0)}_{proposal}$ are determined, the proposal is tuned by adjusting γ_{DRAM} and s_d in the DRAM algorithm as described in part 1 of this project. Here i use the notation γ_{DRAM} for the tuning knob inside DRAM, and γ for the parameter of the SIR model we are trying to infer The covariance of the accepted samples is updated at each iteration and scaled by s_d to get the covariance matrix of the first proposal, and if the first proposal is rejected, the covariance matrix is scaled further by γ_{DRAM} for the second proposal.

Since the ideal acceptance ratio is 20-30%, I try to change s_d and γ_{DRAM} to get an acceptance ratio in this range. While doing this, I also monitor the Marginals, Autocorrelation vs lag plot, the Integrated Autocorrelation(IAC) and the visual inspection of the samples to ensure that the samples are mixing well and the proposal is tuned correctly. Specifically

- Acceptance ratio:
  - if low, decrease s_d and γ_{DRAM}
  - if high, increase s_d and γ_{DRAM}
- Marginals:
  - if clustering around a few points and not exploring enough, increase s_d and γ_{DRAM}
- Autocorrelation:
  - if doesnt decay quickly, increase s_d and γ_{DRAM}
- IAC:
  - if high, increase s_d and γ_{DRAM}

For the sake of speed, to increase the acceptance ratio I preferred to decrease s_d before I increased γ_{DRAM}, because if the first proposal is rejected, the trajectory has to be propagated again for the second proposal, which is computationally expensive. Hence I prefered the first proposal to be accepted more often rather than it rejecting and the 2nd proposal being accepted. γ_{DRAM} is meant to be used when the first proposal rejected for being too exploratory, so it should ideally be ≤ 1 so the second proposal is more conservative.

After tuning, I ended up with $s_d = 9$ and $\gamma_{DRAM} = 0.9$ for the final proposal. I started off with a high acceptance ratio, so I increased s_d to get the acceptance ratio in the desired range. I also increased γ_{DRAM} slightly. I made sure that all other metrics remained good while this happened.

4/5 Analyze your results using the same deliverables as you used in Section 1.

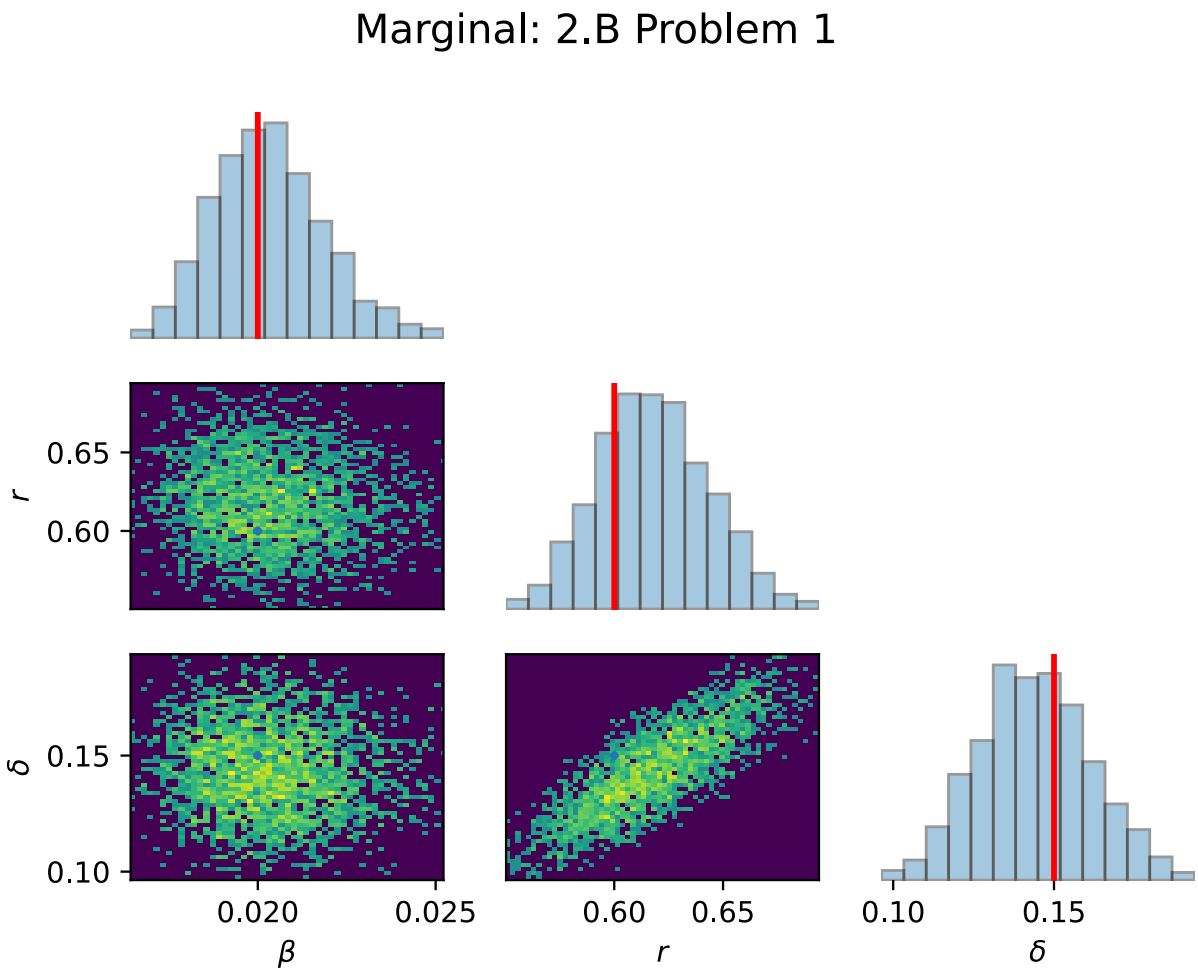

Figure 22: Marginals for Problem 1

- Integrated autocorrelation values:
  - β: 2.60
  - r: 4.61
  - δ: 6.47
- Acceptance ratio: 30%

The acceptance ratio started off very high, but was tuned to 30% while making sure that the samples are mixing well. We can see that the marginals look good, and the Autocorrelation plot decays quickly leading the low values of IAC. Finally, the visual inspection of the samples shows that the samples do not get stuck in a local area and the MC is exploring the parameter space well.

When we compare the value of the true (β, r, δ) with the marginals, we can see that the true values are actually quite close, but not exactly the peak of the gaussian marginals except for β. This could be due to there being noise in our data. The fact that our prior was not very far of from the true values should have helped them get this close, closer than for problem 2A. Given these limitation, we cannot expect the posterior to converge exactly to the true values, but we can see that true values are not that far off from the peak of the marginals, which is a good sign. Also given that all the covariance plots look like gaussians, it makes sense that we have converged close to the true values using a gaussian proposal for our MCMC.

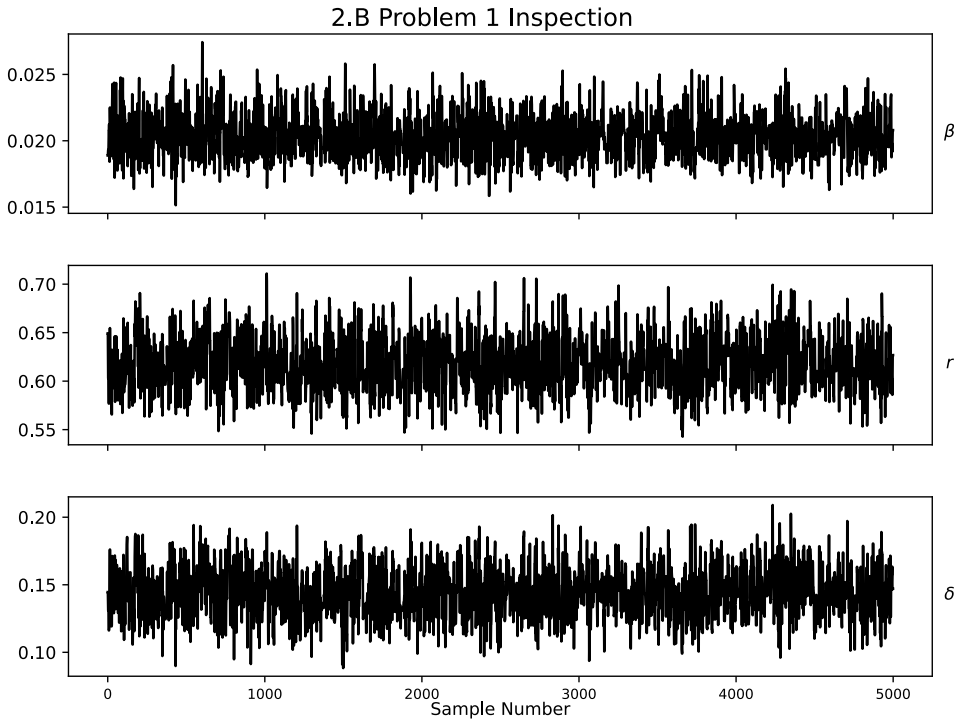

Figure 23: Visual Inspection for Problem 1

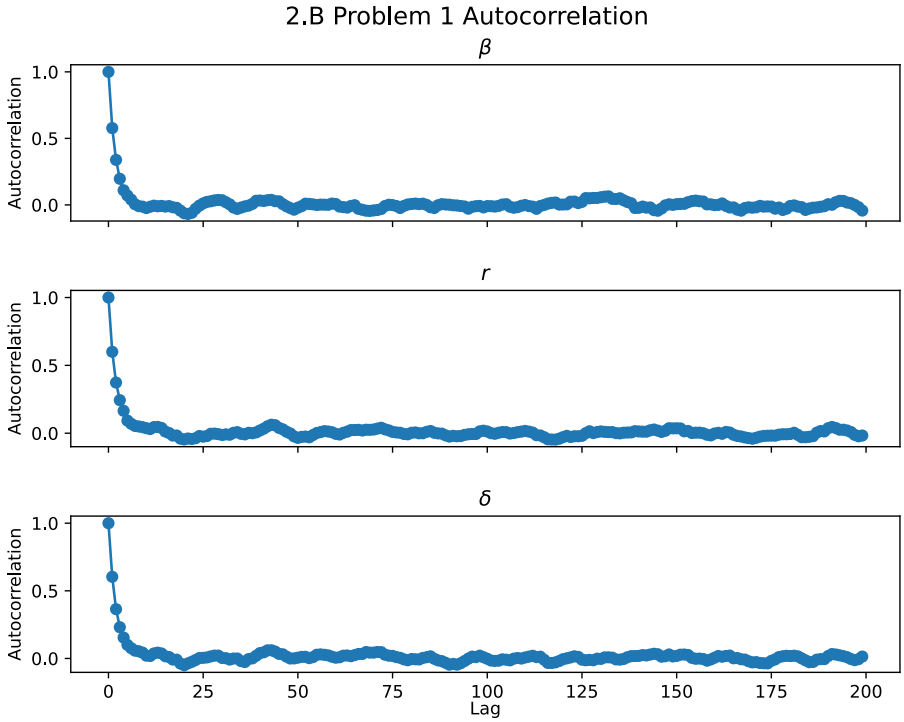

Figure 24: Autocorrelation for Problem 1

6 Plot the prior and posterior predictives of the dynamics (separately)

The true data is in blue and the predicted data is in grey

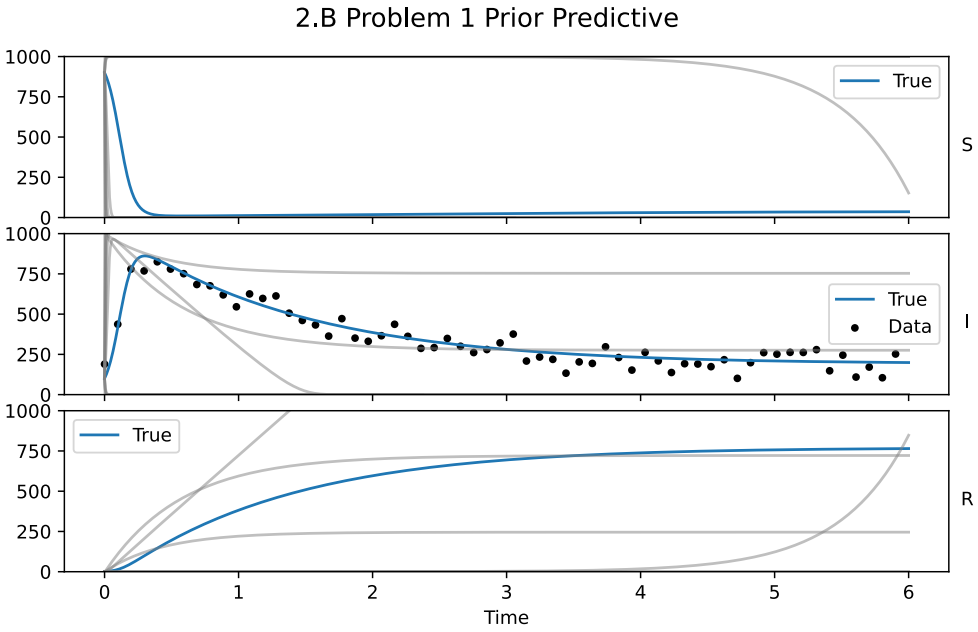

Figure 25: Prior Predictive for Problem 1

The prior predictive looks like it grows/decays exponentially, only being somewhere near the true trajectory in the beginning while the posterior predictive actually matches the true trajectory of the evolution quite closely. This is a good sign that our model is working well and the parameters are being inferred correctly. The difference between the prior and posterior predictives is quite stark, showing the power of Bayesian inference given that after learning the parameters we can make much better predictions.

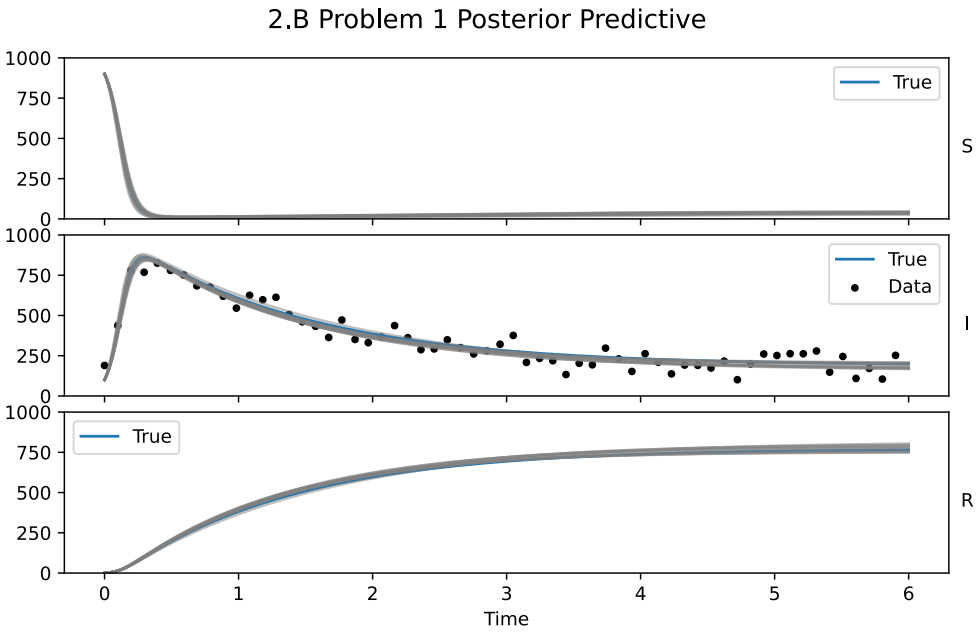

Figure 26: Posterior Predictive for Problem 1

Problem 2 (Not-identifiable): Parameters are $\theta = (\gamma, \kappa, r, \delta)$

1 What is the likelihood model? Please describe how you determine this.

For this exercise, the likelihood model will tell us how likely/probable the observed data is given the parameters of the model $(\gamma, \kappa, r, \delta)$. We only have access to a subset of all data (only I from S, I and R) that the evolution trajectory generates, so we will focus on just the data that is available. Given a function $f(\theta, t)$ that evolves the trajectory of the disease model using the parameters $\theta = (\gamma, \kappa, r, \delta)$ and outputs the Infected $(I^{(t)})$ at any given time t,

Let

$$\begin{aligned} \text{params: } & \theta = (\gamma, \kappa, r, \delta) \\ \text{time instances: } & t = 1, 2, ..., 61 \\ \text{trajectory function: } & f(\theta, t) = I^{(t)} \end{aligned}$$

We have already observed the data $y^{(t)}$ for the true trajectory at each time instance t.

$$\text{actual observed data at time t: } y^{(t)}$$

Hence, the likelihood can be written as:

$$\text{Likelihood: } \mathcal{L}(\theta) = \mathbb{P}(y|\theta)$$

We observe data (I) corrupted by independent Gaussian noise with 0 mean and standard deviation of 50. This means the data observed of the actual disease trajectory is not deterministic, but stochastic and follows a probability distribution. The meaning of $\mathbb{P}(y|\theta)$ is that we will be comparing the observed data $y^{(t)}$ with the trajectory output $I^{(t)}$ by the function $f(\theta, t)$ at each time instance t.

$$\begin{aligned} \text{data noise: } & \xi \sim \mathcal{N}(0, 50^2) \\ \text{distribution of observed data at time t: } & o^{(t)} = I^{(t)} + \xi \end{aligned}$$

$o^{(t)}$ is a gaussian distribution with mean $I^{(t)}$ and variance 50^2.

$$o^{(t)} = \mathcal{N}(I^{(t)}, \Sigma)$$

Now we know that the likelihood for a certain observation of the infected population is determined by how far away it is from the infected population of the trajectory evolved by certain θ, taking into account the noise of in our observations. Different parameters will result in different evolutions, and the observed data might be closer to some evolutions than others. The likelihood then is the probability of the observed data given a trajectory evolved by the chosen parameters. In other words, we are trying to see how probable/likely the data of the infected population is given its noise, if a certain trajectory was evolved by the chosen parameters. The trajectory determined by the parameters is in continuous time, while the observed data is in discrete time. We need to select the infected population of the evolution at the same time instances as the observed data. The probability of $y^{(t)}$ given θ is then the probability of the gaussian distribution with mean $I^{(t)}$ (output of the function $f(\theta, t)$) and variance 50^2 at $y^{(t)}$. The likelihood is then the product of the probabilities of the $y^{(t)}$ given $o^{(t)}$ at each time instance t.

$$\text{Likelihood: } \mathcal{L}(\theta) = \prod_{t=1}^{61} \mathbb{P}(y^{(t)}|o^{(t)})$$

For ease of computation, we will actually use the log-likelihood.

$$\text{Log-Likelihood: } \log \mathcal{L}(\theta) = \sum_{t=1}^{61} \log \mathbb{P}(y^{(t)}|o^{(t)})$$

Since the noise is gaussian, we can easily use log probability of a gaussian to compute this value.

Let

$$\sigma = 50$$

Then

$$\begin{aligned}
\log \mathbb{P}(y^{(t)}|o^{(t)}) &= \log \mathcal{N}(y^{(t)}|I^{(t)}, \Sigma) \\
&= \log \left(\frac{\exp\left(-\frac{1}{2}(\frac{y^{(t)} - I^{(t)}}{\sigma})^2\right)}{\sigma\sqrt{2\pi}} \right) \\
&= -\frac{1}{2}\log(2\pi) - \log(\sigma) - \frac{1}{2}\left(\frac{y^{(t)} - I^{(t)}}{\sigma}\right)^2
\end{aligned}$$

We can then plug this into the log-likelihood function to compute the log-likelihood of the data given the parameters for all time instances.

2 What is the form of the posterior?

In Bayesian inference, the posterior is the probability of the parameters($\theta = \gamma, \kappa, r, \delta$) given the observed data(y). It is proportional to the likelihood of the data given the parameters and the prior probability of the parameters.

$$\mathbb{P}(\theta|y) = \frac{\mathbb{P}(\theta)\mathbb{P}(y|\theta)}{\mathbb{P}(y)}$$

Where

$$\begin{aligned}
\mathbb{P}(\theta) &= \text{Prior probability of the parameters} \\
\mathbb{P}(y|\theta) &= \text{Likelihood of the data given the parameters} \\
\mathbb{P}(y) &= \text{Marginal likelihood of the data} \\
\mathbb{P}(\theta|y) &= \text{Posterior probability of the parameters given the data}
\end{aligned}$$

The posterior is basically the prediction of our parameters $\theta = (\gamma, \kappa, r, \delta)$ using our prior belief about $(\gamma, \kappa, r, \delta)$ (Prior), how probable the data y is given a trajectory propagated by the parameters $(\gamma, \kappa, r, \delta)$ (Likelihood) and the probability of the data y itself(Marginal Likelihood). The Prior for each θ_i is given in the problem statement to be $\theta_i \sim \mathcal{N}(0, 1)$, so we can both sample from it and determine the probability density. The Likelihood can be computed as described in the previous section. The Marginal Likelihood is the probability of the data y given the SIR model, which is the integral of its probability over all possible parameters $(\gamma, \kappa, r, \delta)$. Since the space of parameters is unbounded unsolvable analytically, this integral is intractable. This is why we use MCMC methods to sample from the posterior distribution directly, because it is able to sample from a distribution that cannot be be evaluated up to their normalizing constant, which is the marginal likelihood of the data in this case.

In this way, we can compute the posterior and log posterior distribution by:

$$\begin{aligned}
\text{Posterior: } \mathbb{P}(\theta|y) &: \mathbb{P}(\theta)\mathbb{P}(y|\theta) \\
\text{Log Posterior: } \log \mathbb{P}(\theta|y) &: \log \mathbb{P}(\theta) + \log \mathbb{P}(y|\theta)
\end{aligned}$$

Here the prior is a 4-D multivariate gaussian but with an identity covariance, which means we can treat all the parameters are independent and calculate the logPDF for each parameter independently.The likelihood is a 1-D gaussian as well, as described in the previous section and can be calculated as described in the previous section.

3 How did you tune your proposal? Think carefully about what a good initial point and initial covariance could be?

The proposal used for the DRAM is a gaussian random walk proposal. For the initial point we can start off from the Maximum Aposteriori(MAP) estimate of the posterior distribution. This will be a certain value of $\theta = (\gamma, \kappa, r, \delta)$. The MAP estimate can be found by maximizing the logPDF of the posterior distribution. If we simplify the problem and approximate the posterior as a 4D gaussian, the scaled negative inverse of the hessian at the MAP can be used as the covariance matrix for a random walk proposal. This is also called the Laplace Approximation. The scaling factor of the negative inverse hessian is important for determining the initial covariance of the the random walk proposal and has been taken from the suggestion in the notes (ref: eq 16.6). Specifically:

Let

$$\text{Log Posterior: } \log \mathbb{P}(\theta|y) = f_{Xpost}$$

Then

$$\begin{aligned}
\theta_{MAP} &= \arg\max_x f_{Xpost} \\
\theta^{(0)} &= \theta_{MAP} \\
\Sigma^{(0)}_{proposal} &= -\frac{2.4^2}{d}\left(\nabla^2 f_{Xpost}(\theta_{MAP})\right)^{-1}
\end{aligned}$$

Using `scipy.optimize.minimize` with $-f_{Xpost}$ as the objective function, we find the MAP and initial proposal covariance matrix:

- $\theta^{(0)} = (\gamma, \kappa, r, \delta)$: (0.14170413, 0.14170406, 0.61444686, 0.14098853)
- $$\Sigma^{(0)}_{proposal}: \begin{bmatrix} 0.00076125 & -0.00074788 & 0.00018916 & 0.00017149 \\ -0.00074788 & 0.00097256 & -0.00019211 & -0.00017223 \\ 0.00018916 & -0.00019211 & 0.00070551 & 0.00037287 \\ 0.00017149 & -0.00017223 * 0.00037287 & 0.00033792 & \end{bmatrix}$$

After the initial $\theta^{(0)} = (\gamma^{(0)}, \kappa^{(0)}, r^{(0)}, \delta^{(0)})$ and $\Sigma^{(0)}_{proposal}$ are determined, the proposal is tuned by adjusting γ_{DRAM} and s_d in the DRAM algorithm as described in part 1 of this project. Here i use the notation γ_{DRAM} for the tuning knob inside DRAM, and γ for the parameter of the SIR model we are trying to infer. The covariance of the accepted samples is updated at each iteration and scaled by s_d to get the covariance matrix of the first proposal, and if the first proposal is rejected, the covariance matrix is scaled further by γ_{DRAM} for the second proposal.

Since the ideal acceptance ratio is 20-30%, I try to change s_d and γ_{DRAM} to get an acceptance ratio in this range. While doing this, I also monitor the Marginals, Autocorrelation vs lag plot, the Integrated Autocorrelation(IAC) and the visual inspection of the samples to ensure that the samples are mixing well and the proposal is tuned correctly. Specifically

- Acceptance ratio:
  - if low, increase s_d and γ_{DRAM}
  - if high, decrease s_d and γ_{DRAM}
- Marginals:
  - if clustering around a few points and not exploring enough, increase s_d and γ_{DRAM}
- Autocorrelation:
  - if doesnt decay quickly, increase s_d and γ_{DRAM}
- IAC:
  - if high, increase s_d and γ_{DRAM}

For the sake of speed, to increase the acceptance ratio I preferred to decrease s_d before I increased γ_{DRAM}, because if the first proposal is rejected, the trajectory has to be propagated again for the second proposal, which is computationally expensive. Hence I prefered the first proposal to be accepted more often rather than it rejecting and the 2nd proposal being accepted.

After tuning, I ended up with $s_d = 2$ and $\gamma_{DRAM} = 0.5$ for the final proposal.

For this specific problem, I started off with a really low acceptance ratio, but extremely strongly clustered points in the visual inspection along with slowly decaying Autocorrelation and high IAC values. Thus, I was in a dilemma, whether to increase acceptance ratio by reducing s_d or if i should try and get the DRAM algorithm to explore more and lead to less clustered points. In the end, I decided to look at the posterior predictives and select the tuning parameters that gave me answers that got close to the true evolution of the disease. This makes sense to me because ultimately we are interested in seeing if our inferred θ can accurately reproduce the original evolution of the disease.

4/5 Analyze your results using the same deliverables as you used in Section 1.

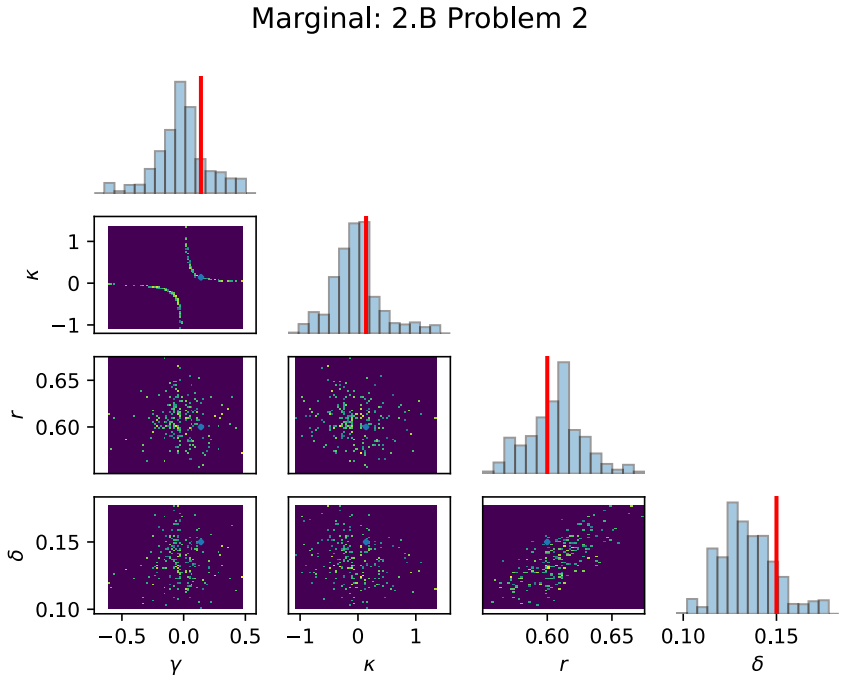

Figure 27: Marginals for Problem 1

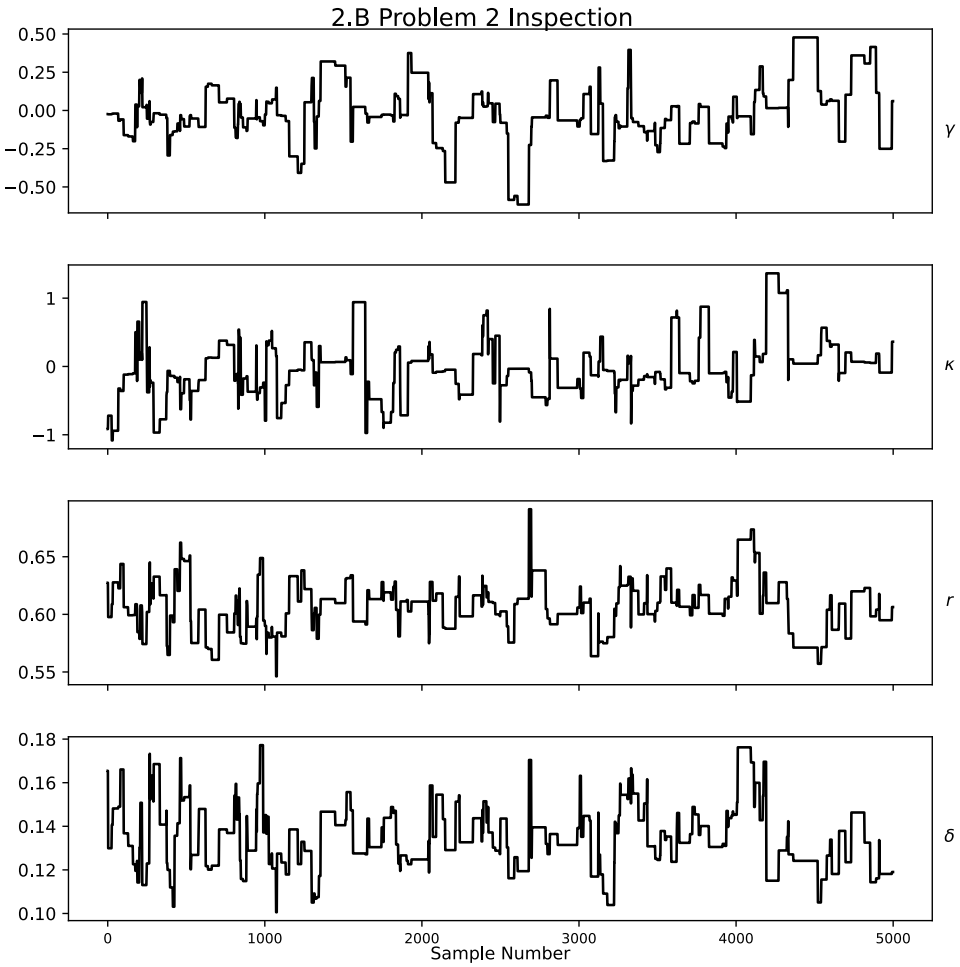

Figure 28: Visual Inspection for Problem 1

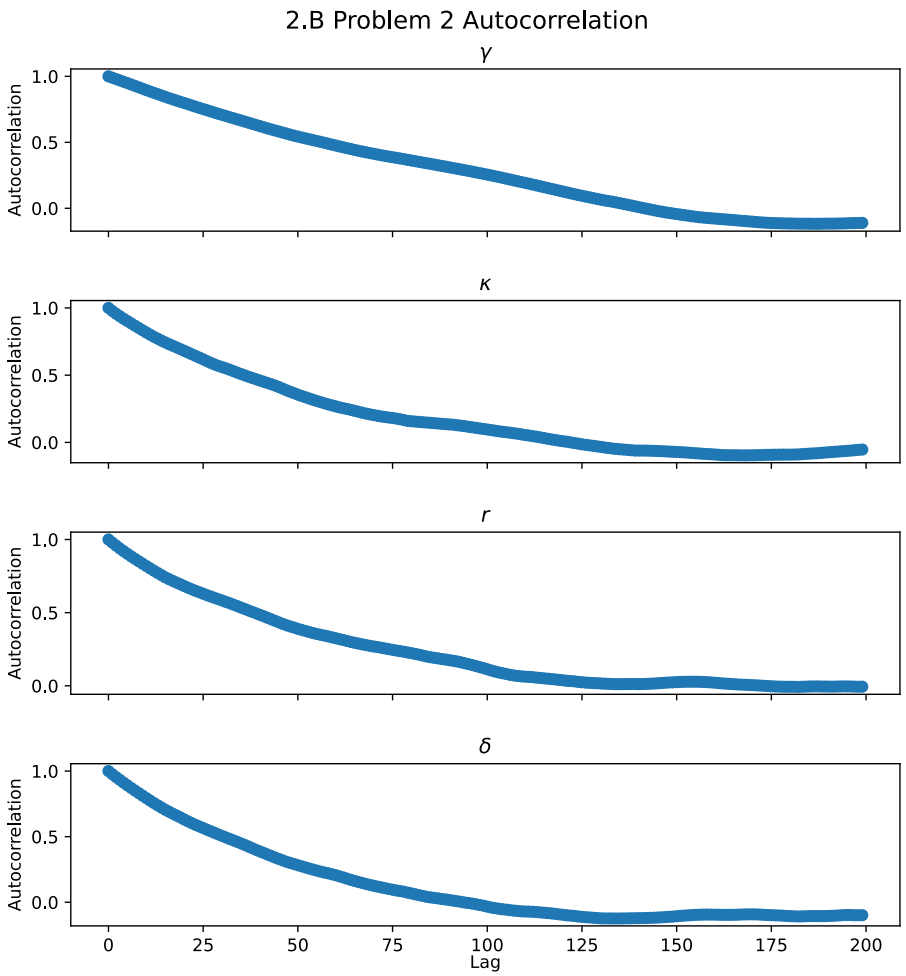

Figure 29: Autocorrelation for Problem 1

- Integrated autocorrelation values:
  - γ: 621
  - κ: 333
  - r: 121
  - δ: 78
- Acceptance ratio: 4%

The acceptance ratio for this problem started off very low, and I tried to increase it by decreasing s_d, but that caused the visual inspection plot to look even more clustered. Here we can see that all the metrics are pretty bad, but that is because I prioritized getting the posterior predictive to match the true evolution of the disease as closely as possible

When we compare the value of the true parameters with the marginals, we can see that the marginals are not exactly gaussian, especially when we look at the one for γ and κ, As discussed before, it looks like the graph of $\frac{1}{x}$, which makes sense since both of them multiplied together should be similar to β. This problem does not just stem from the fact that we have noise in the data, but also the fact that our proposal would have been wildly different from the actual distribution of the data as we can see in the marginals.

6 Plot the prior and posterior predictives of the dynamics (separately)

The true data is in blue and the predicted data is in grey

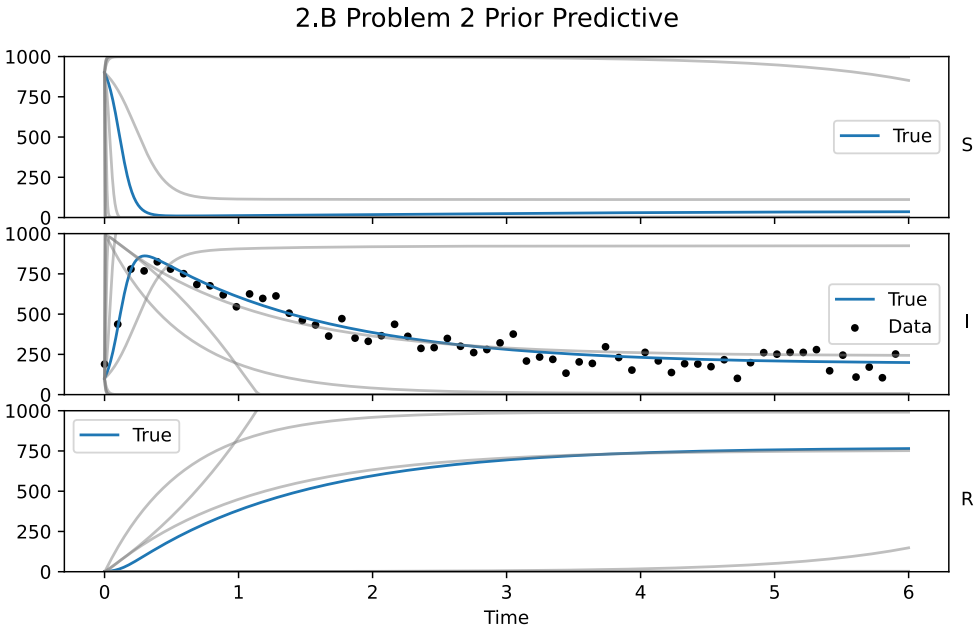

Figure 30: Prior Predictive for Problem 2

The prior predictive looks totally random, only being close to the true trajectory in the beginning while the posterior predictive actually matches the true trajectory of the evolution quite closely. This is a good sign that our model is working well and the parameters are being inferred correctly. The difference between the prior and posterior predictives is quite stark, showing the power of Bayesian inference given that after learning the parameters we can make much better predictions.

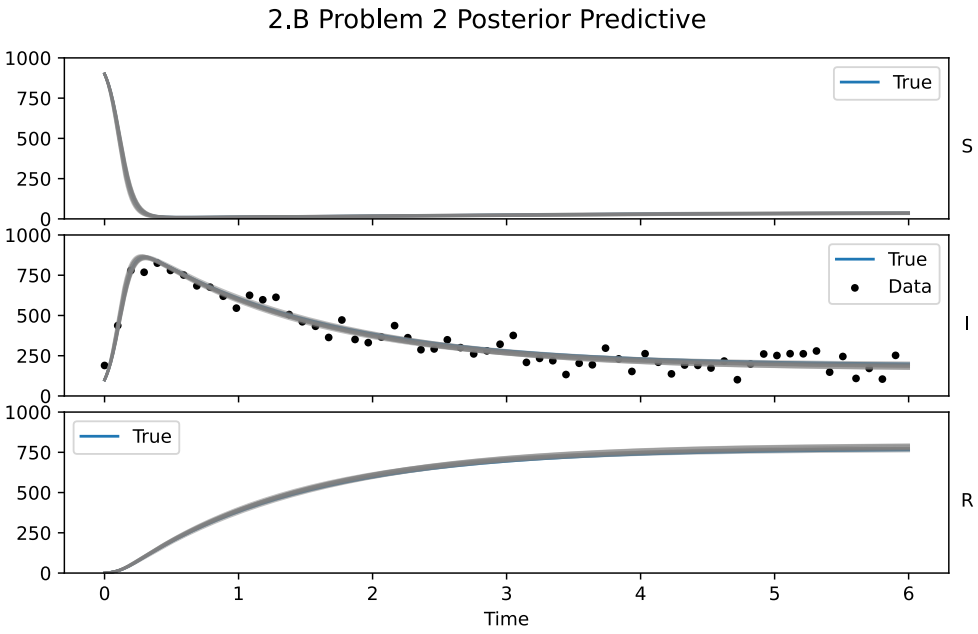

Figure 31: Posterior Predictive for Problem 2

Please comment on the following:

- How does non-identifiability affect the Bayesian approach?

The non-identifiability makes it extremely difficult to infer the parameters, especially because the parameters γ and κ related to β like $\beta = \gamma\kappa$, which makes the posterior distribution look like $\frac{1}{x}$. We have extremely low acceptance ratio, with seemingly no way to tune it without losing out in other metrics to analyse the DRAM MCMC

- In many models it may not be clear by inspection that certain parameters are non-identifiable. How can you use the Bayesian approach to probe whether this might be the case?

In such models, we can see if our analysis metrics like the ones discussed in this report go against each other in terms of tuning the MCMC. If they do, it is highly likely that the parameters are non-identifiable. We can also try to have an approach where we fix one of the parameters and only search for the others, and doing this sequentially might help us understand the non-identifiability of the parameters.