Oden Institute REPORT 20-10

May 2020

Assessment of Predictability of a Class of Models of Growth of Coronavirus 19 Cases as an Exercise in Predictive Computational Science

by

J. Tinsley Oden, Prashant K. Jha, Lianghao Cao, Taemin Heo, Jing Hu, Mathew Hu, Jonathan Kelley, Jaime D. Mora Paz, Cyrus Neary, Akhil Potla, Sheroze Sherriffdeen, Chase Tessmer, and Christine Yang

Oden Institute for Computational Engineering and Sciences
The University of Texas at Austin
Austin, Texas 78712

Reference: J. Tinsley Oden, Prashant K. Jha, Lianghao Cao, Taemin Heo, Jing Hu, Mathew Hu, Jonathan Kelley, Jaime D. Mora Paz, Cyrus Neary, Akhil Potla, Sheroze Sherriffdeen, Chase Tessmer, and Christine Yang, "Assessment of Predictability of a Class of Models of Growth of Coronavirus 19 Cases as an Exercise in Predictive Computational Science ," Oden Institute REPORT 20-10, Oden Institute for Computational Engineering and Sciences, The University of Texas at Austin, May 2020.

ASSESSMENT OF PREDICTABILITY OF A CLASS OF MODELS OF GROWTH OF CORONAVIRUS 19 CASES AS AN EXERCISE IN PREDICTIVE COMPUTATIONAL SCIENCE

J. Tinsley Oden, Prashant K. Jha, Lianghao Cao, Taemin Heo,
Jing Hu, Mathew Hu, Jonathan Kelley, Jaime D. Mora Paz, Cyrus Neary,
Akhil Potla, Sheroze Sherriffdeen, Chase Tessmer, Christine Yang

PREFACE

In the Spring Semester of 2020, the class CSE 397: Foundations of Predictive Computational Science, jointly listed as EM 397, was taught within the CSEM graduate program at University of Texas at Austin by JTO and Postdoctoral Assistant Prashant Jha (via Zoom). As a final assignment, the class was asked to use the Bayesian-based methods for model calibration, validation, parameter estimation, and prediction of quantities of interest in the presence of uncertainties covered in class to study the predictability of a simple NIH growth model of cases of a disease in a pandemic such as COVID-19. Data for COVID-19 cases from the US, Japan, and South Korea were available on the internet and were used as data. Prashant Jha also worked out an analysis using these data and made some of his calculations available to the class.

This report contains the solutions provided by the class. Different metrics and tolerances for validation may be found among these results as well as different measures used to qualify uncertainty in quantities of interest, generally taken to be the total number of cases predicted to exist after around 100 days based on calibration data, over around 40 days, and validation data for a period of days after that. We believe these calculations provide very good examples of the steps that can be used to make meaningful scientific predictions in the presence of uncertainty for a very relevant and important class of biomedical phenomena. We remark that the range of QoI predictions suggests that the simple model employed may be incapable of capturing the actual trends in this pandemic. Nevertheless, the general process of quantifying predictability is well demonstrated in various solutions.

The authors thank Charlott Low for organizing these assigned solutions into chapters and compiling them into a report suitable for for publication.

TABLE OF CONTENTS

Chapter 1

Gentle introduction to Bayesian calibration, validation, and prediction with application to COVID-19 data

Prashant K. Jha, J. Tinsley Oden

Oden Institute for Computational Engineering and Sciences, University of Texas at Austin, TX

Abstract

We apply the Bayesian techniques of the model calibration, validation, and prediction to the real world data. We consider the confirmed cases of COVID-19 in Japan as the data and calibrate and validate the generalized (sub-exponential) growth model. We use the calibrated model to make a prediction at 100^{th} day and quantify the uncertainty in the prediction.

Keywords: Bayesian statistics, model calibration, prediction, quantity of interest

1. Introduction

Consider a transient scalar field $C : [0, T] \to [0, \infty)$ denoting the number of confirmed cases of COVID-19 virus as a function of time (in units of days) in some specific region, say Japan. We consider a simple generalized growth model for C. Following [5], we assume that C satisfies following ODE:

$$\frac{dC}{dt} = rC(t)^p, \qquad \forall t \in (0, T], \tag{1}$$

where $r \geq 0$ is the growth rate and $p \in [0, 1]$ is the deceleration of growth. When $p = 0$, the model is a linear growth model, and when $p = 1$, the model is an exponential growth model. Given the initial condition, i.e. the number of COVID-19 cases at $t = 0$, C_0, (1) can be solved to get, for all $t \in [0, T]$,

$$C(t) = \begin{cases} \left(\frac{r}{m}t + (C_0)^{1/m}\right)^m, & \text{when } p \in (0, 1), \\ C_0 + rt, & \text{when } p = 0, \\ C_o \exp[rt], & \text{when } p = 1, \end{cases} \tag{2}$$

Model, for different parameters (r, p), is depicted in Fig. 1.

Objective of this work is to introduce the Bayesian techniques of model calibration, validation, and prediction by applying it to the present day relevant problem. We consider Japan's COVID-19 cases as the given data to fit the model (2) using Bayesian approach. We highlight and discuss various steps in the Bayesian approach including the practical challenges occurring during the numerical implementation.

We seek to find the optimal value of parameters (r, p) in the model (2) that best describes the real COVID-19 data. In other words, we want to fit (2) to the COVID-19 data. In the Bayesian approach, instead of finding one fixed value of parameter (r, p), we seek the joint probability distribution of (r, p). Using the joint probability distribution, we not only make the prediction about the quantity of interest (QoI), we can also measure the uncertainty in the prediction (or confidence in the prediction). In the remainder of this section, we present a brief overview of Bayesian learning in the context of fitting model (2) to the COVID-19 data. Interested readers can find more details in the reference [4] and references therein.

Email addresses: pjha@utexas.edu (Prashant K. Jha), oden@oden.utexas.edu (J. Tinsley Oden)

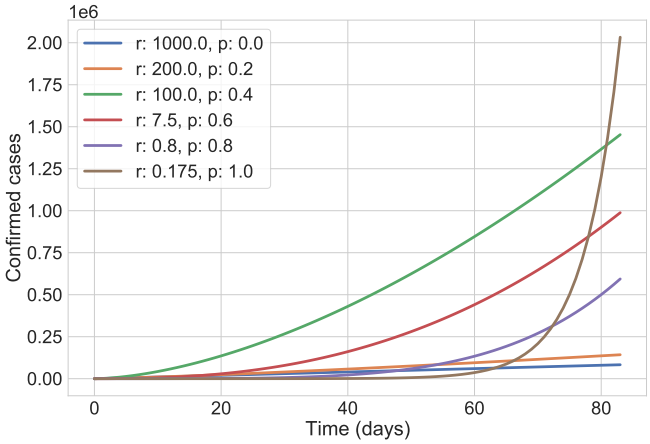


Figure 1: **Plot of the generalized growth model for different model parameters.**

Let $\theta = (r, p) \in \Theta := [0, \infty) \times [0, 1]$ denote the model parameter where Θ is the space where the parameters live in. Let $I = [0, T]$ be the time interval of interest, where T is the final time at which we want to get the model prediction. We denote the discrete data $Y = (Y_1, Y_2, .., Y_N)$ where Y_i is the total number of COVID-19 cases in specific region at day i. Let $C(t; \theta)$ denote the model output, i.e. the total number of confirmed cases, at time t for parameter θ.

Suppose $\pi(\theta)$ denotes the prior probability distribution of the model parameter, $\pi(Y|\theta)$ denotes the conditional probability of the data when the parameter is fixed to θ (also called the likelihood function), $\pi(\theta|Y)$ denotes the conditional probability of the parameter for a given data Y (also called the posterior), and $\pi(Y)$ denotes the evidence. We apply the Bayes' rule to relate the posterior (unknown) to the likelihood (known) and the prior (assumed):

$$\pi(\theta|Y) = \frac{\pi(Y|\theta)\pi(\theta)}{\pi(Y)}, \tag{3}$$

where evidence $\pi(Y)$ is the marginalization of the numerator in (3) (so that the posterior is integrated to 1). It is given by

$$\pi(Y) = \int_{\Theta} \pi(Y|\theta)\pi(\theta)d\theta. \tag{4}$$

We see that, up to a proportionality constant, posterior is proportional to the multiple of likelihood function and prior distribution. Once the definition of the prior and the likelihood function is fixed, we can generate a parameter samples according to the posterior distribution using one of the Markov chain Monte Carlo methods (more details below).

1.1. Prior distribution

We assume no prior knowledge about the parameters (r, p). Therefore, we define the prior distribution as a uniform distribution over Θ, i.e.,

$$\pi(\theta) = \chi_{\Theta}(\theta), \tag{5}$$

where χ_{Θ} is the usual indicator function.

1.2. Noise in the data and the model inadequacy

1.2.1. Experimental noise

The data Y can be noisy, that is, the real number of COVID-19 cases could be bit different from the recorded data Y. This is called the experimental noise. Suppose g is the real data, Y is the recorded data with some margin of error, and ϵ is the noise, then Y must be related to g by some function f (unknown)

$$Y = f(g, \epsilon).$$

To proceed further, we need to assume some reasonable form of this function f. We suppose that ϵ follows the Gaussian distribution with mean 0, take $f(g, \epsilon) = g + \epsilon$ (additive noise). This results in

$$Y = g + \epsilon. \tag{6}$$

Since $Y = (Y_1, ..., Y_N)$ is a vector, we can assume $\epsilon = (\epsilon_1, .., \epsilon_N)$ where each ϵ_i is given by the Gaussian distribution with 0 mean and σ_i std deviation.

1.2.2. Model inadequacy

For fixed parameter θ, the model output at t_i is denoted as $C(t_i; \theta)$. Let $\bar{C}(\theta) := (C(t_i; \theta), ..., C(t_N; \theta))$ is the vector of model output. Assuming that the model is imperfect, we introduce an additive modeling error $\gamma(\theta)$ such that the real data and the model output $\bar{C}(\theta)$ are related to each other by:

$$g - \bar{C}(\theta) = \gamma(\theta). \tag{7}$$

γ can be assumed to follow some simpler probability distribution. As we see next, we can combine the experimental noise and the model inadequacy, and assume the probability distribution for the combined error $\gamma + \epsilon$.

Following [Section 4.3, [4]], and also see (6), we have

$$Y - \epsilon - \bar{C}(\theta) = \gamma(\theta) \Rightarrow Y - \bar{C}(\theta) = \epsilon + \gamma(\theta), \tag{8}$$

i.e., the difference between the recorded data and the model output is equal to the sum of the noise and the model inadequacy.

In rest of this article, we assume $\epsilon + \gamma(\theta) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\sigma})$, where $\boldsymbol{\sigma}$ is the $N \times N$ diagonal matrix with $\boldsymbol{\sigma}_{ii} = \sigma$ for $1 \leq i \leq N$. Here $\theta \sim \mathcal{N}(\mu, \sigma)$ means that θ is sampled from Gaussian distribution or the θ is the random variable with the probability distribution given by $\mathcal{N}(\mu, \sigma)$.

1.3. Likelihood function

Recall that $\bar{C}(\theta) = (C(t_1; \theta), ..., C(t_N; \theta))$ is a vector of model output at times $(t_1, ..., t_N)$ associated to the data Y. We define the likelihood function as follows:

$$\begin{aligned} \pi(Y|\theta) &= \mathcal{N}(Y - \bar{C}(\theta), \boldsymbol{\sigma}) \\ &= \prod_{i=1}^{N} \frac{1}{\sigma_{ii}\sqrt{2\pi}} \exp\left[-\frac{|Y_i - C(t_i; \theta)|^2}{2\sigma_{ii}^2}\right], \end{aligned} \tag{9}$$

where we have assumed $\boldsymbol{\sigma}$ to be the diagonal matrix with $\sigma_{ii} = \sigma$ for $1 \leq i \leq N$. We can also compute the logarithm of the likelihood function:

$$\log(\pi(Y|\theta)) = \sum_{i=1}^{N} \left[-\frac{|Y_i - C(t_i; \theta)|^2}{2\sigma_{ii}^2} - \log(\sigma_{ii}\sqrt{2\pi})\right]. \tag{10}$$

1.4. Posterior distribution

With the definitions of the likelihood function and the prior distribution at hand, we apply the Bayes' rule to get the posterior distribution:

$$\pi(\theta|Y) \propto \pi(Y|\theta)\pi(\theta). \tag{11}$$

One can change the proportionality sign to equality by introducing the evidence $\pi(Y)$, see (3).

1.5. Quantity of interest

Once the model is calibrated and validated, we use the posterior distribution for the model parameters to compute the quantity of interest (QoI). Suppose $Q(t, C)$ is the quantity of interest which is a function of time t and function of model output C. With the posterior $\pi(\theta|Y)$, the quantity of interest Q in the Bayesian approach is a random field as

$$Q = Q(t, C) = Q(t, C(\theta)) \tag{12}$$

is a function of θ through model C, and θ is the random field with the probability distribution $\pi(\theta|Y)$. The mean of QoI is given by

$$\mathbb{E}[Q](t, C) = \int_{\Theta} Q(t, C(\theta))\pi(\theta|Y)d\theta. \tag{13}$$

The uncertainty in the predicted QoI is given by the standard deviation of Q, i.e.,

$$\mathbb{V}[Q](t, C) = \int_{\Theta} (Q(t, C(\theta)) - \mathbb{E}[Q](t, C))^2\pi(\theta|Y)d\theta. \tag{14}$$

1.6. Markov chain Monte Carlo simulation

While (11) relates the known distributions, the likelihood function and the prior distribution, to the desired posterior distribution, we still need a method to find the posterior distribution using (11).

Let f and g are two probability distributions such that $f \propto g$. The idea is to use the known distribution g to compute the target distribution f. This is achieved by methods collectively known as Markov chain Monte Carlo. In this work, we use the Metropolis-Hastings method [1]. We consider a symmetric proposal distribution $q(\theta'|\theta) \sim \mathcal{N}(\theta, \boldsymbol{\sigma}_p)$ where θ' is the new sample point, θ is the current sample point, and $\boldsymbol{\sigma}_p$ is the covariance matrix. From (11), we have a target distribution $f = \pi(\theta|Y)$ and the known probability distribution $g = \pi(Y|\theta)\pi(\theta)$. The acceptance distribution $\alpha(\theta', \theta)$, which gives the probability of accepting sample θ' given that we are at sample θ, is defined as:

$$\alpha(\theta', \theta) = \frac{q(\theta|\theta')g(\theta')}{q(\theta'|\theta)g(\theta)} = \frac{g(\theta')}{g(\theta)}, \tag{15}$$

where we used the fact that q is symmetric and so $q(x|y) = q(y|x)$. If $\alpha \geq 1$ then we accept the sample θ' and if $\alpha < 1$ then we reject the sample θ' with probability α. The MH (Metropolis-Hastings) algorithm is presented in Alg. 1.

1.7. Posterior distribution from the MCMC samples

One last practical aspect of the Bayesian approach is how one can sample from the accepted samples of MCMC method or in other words what is the probability distribution inherent in the accepted MCMC samples? This question arises when we go from the calibration step to the validation step. There are two methods to obtain the probability distribution from the MCMC samples:

- *Gaussian approximation:* Compute the mean μ_{sample} and the standard deviation σ_{sample} of the samples and approximate the posterior by Gaussian distribution, $\pi(\theta|Y) \sim \mathcal{N}(\mu_{sample}, \sigma_{sample})$.
- *Kernel density estimation:* A more accurate approach is to apply the kernel density estimation to approximate the posterior.

We will compare the results with above two approximations in Sec. 4.

Algorithm 1 Metropolis-Hastings sampling method

```
# initialize parameter, create vectors to store samples
theta = theta0
accepted = [ ]
rejected = [ ]
# start random walk
for each integer 0 ≤ i ≤ N_iters do
    # new sample point
    theta_new ~ q(·| theta)
    like_prior = likelihood(theta) * prior(theta)
    like_prior_new = likelihood(theta_new) * prior(theta_new)
    alpha = like_prior_new / like_prior
    if alpha ≥ 1 then
        # accept theta_new
        accepted.add(theta_new)
        theta = theta_new
    else
        # reject with probability alpha
        U ~ Uniform(0,1)
        if U ≤ alpha then
            # accept theta_new
            accepted.add(theta_new)
            theta = theta_new
        end if
    end if
end for
return accepted
```

1.8. Calibration, validation, and prediction

Bayesian fitting of the data involves three steps:

- *Calibration:* We divide the data in the calibration set and the validation set. Let $Y^c, \bar{C}^c(\theta), \pi^c(\theta), \pi^c(Y^c|\theta), \pi^c(\theta|Y^c)$ denote the calibration data, model output at times associated to calibration data, prior in the calibration step, likelihood function defined on the calibration data Y^c and model output $\bar{C}^c$, and calibration posterior. The calibration of the model consists of three steps:

  1. Select the prior as follows

  $$\pi^c(\theta) = \chi_\Theta(\theta).$$

  The likelihood $\pi^c(Y^c|\theta)$ is defined according to (9) with the data Y^c instead of Y. The posterior follows the Bayes' rule, i.e., $\pi^c(\theta|Y^c) \propto \pi^c(Y^c|\theta)\pi^c(\theta)$.

  2. Apply MCMC to get the parameter samples following $\pi^c(\theta|Y^c)$ distribution.
  3. Find the approximate posterior distribution $\pi^c(\theta|Y^c)$ from the MCMC samples, using either the Gaussian approximation or the Kernel Density Estimation.

- *Validation:* Let $Y^v, \bar{C}^v(\theta), \pi^v(\theta), \pi^v(Y^v|\theta), \pi^v(\theta|Y^v)$ are the validation data, model output at times corresponding to validation data, prior in the validation step, likelihood function defined on the validation data, and the validation posterior. The steps are as follows:

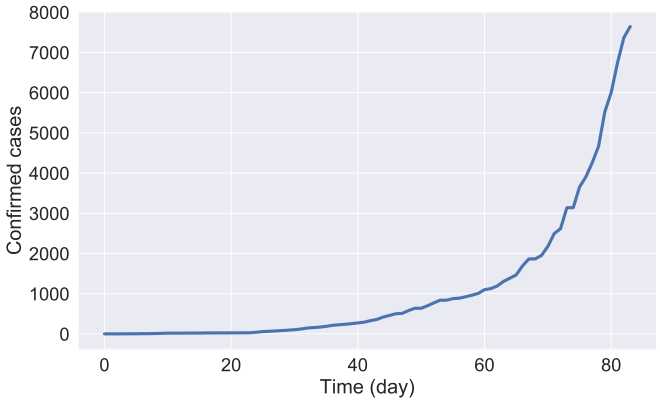


Figure 2: **Confirmed cases in Japan.**

1. Select prior as follows

$$\pi^v(\theta) = \pi^c(\theta|Y^c). \tag{16}$$

The likelihood $\pi^v(Y^v|\theta)$ is defined similar to $\pi^c(Y^c|\theta)$. The posterior satisfies $\pi(\theta|Y^v) \propto \pi^v(Y^v|\theta)\pi^v(\theta)$.

2. Apply MCMC to get the parameter samples following $\pi^v(\theta|Y^v)$ distribution. Suppose the set of samples is denoted as $\Theta^v = (\theta^v_1, ..., \theta^v_{N^v})$.

3. Compute the validation error. Quantity of interest is the number of confirmed cases at validation times $(t^v_1, ..., t^v_{N_{valid}})$. At any given time, the number of confirmed cases is the mean of model ouput over MCMC parameter samples, i.e.

$$Q^v(t, C) := \frac{1}{N^v} \sum_{s=1}^{N^v} C(t; \theta^v_s). \tag{17}$$

We define the validation error e^v as follows:

$$e^v := \frac{\sum_{i=1}^{N_{valid}} |Y^v_i - Q^v(t^v_i, C)|^2}{\sum_{i=1}^{N_{valid}} |Y^v_i|^2}. \tag{18}$$

We consider the model as "Not invalid" if $e^v \leq \gamma_{tol}$. γ_{tol} is the user defined tolerance.

- *Prediction:* If the model is "Not invalid", then the number of confirmed cases $Q = C(T = 100; \theta)$ is the random field. The mean of Q is the model prediction and the standard deviation is the uncertainty in the prediction.

2. COVID-19 data and the hyper parameters

The data for novel coronavirus (COVID-19) is freely available from various sources. Three of these sources are listed in [2]. For the demonstration, we consider the number of confirmed cases recorded in Japan at different days starting from 22 January 2020 until 14 April 2020 (total 84 days), see Fig. 2.

We consider $Y^c = (Y_1, ..., Y_{60})$ at days $t^c = (1, ..., 60)$ as the calibration data and $Y^v = (Y_{61}, ..., Y_{84})$ at days $t^v = (61, ..., 84)$ as the validation data. The final time is $T = 100$. This is also the time at which we want to predict the number of confirmed cases.

2.1. Hyper parameters

We fix the standard deviation σ in Gaussian distribution for the noise and the model inadequacy as $\sigma = 100$.

The proposal distribution $q(\theta'|\theta)$ in MCMC method is assumed to be the Gaussian $\mathcal{N}(\theta, \boldsymbol{\sigma}_p)$ where $\boldsymbol{\sigma}_p$ is the covariance matrix. We consider

$$\boldsymbol{\sigma}_p = \begin{bmatrix} \sigma_{p,11} & 0 \\ 0 & \sigma_{p,22} \end{bmatrix}, \tag{19}$$

where $\sigma_{p,11} = 1$ and $\sigma_{p,22} = 0.01$.

The number of MCMC iteration is fixed to $N_{iter} = 100000$.

2.2. Model selection tolerance

We set $\gamma_{tol} = 0.01$.

3. Calibration step

The initial value of the parameter was taken as $\theta_0 = (0, 0)$. The acceptance rate in the Metropolis-Hastings was 0.481%. The samples generated in the first 20000 iteration (i.e. first 20% of the iterations) were discarded. The mean of the samples was $\theta^c_{mean} = (r^c_{mean}, p^c_{mean}) = (0.305, 0.751)$. The model output using the mean of samples as parameter is shown in Fig. 3.

Histogram plot, see Fig. 4, shows that there is a strong correlation between parameter r and p. In Fig. 5, we show the histogram with the Gaussian and the KDE approximation of the posterior $\pi^c(\theta|Y^c)$.

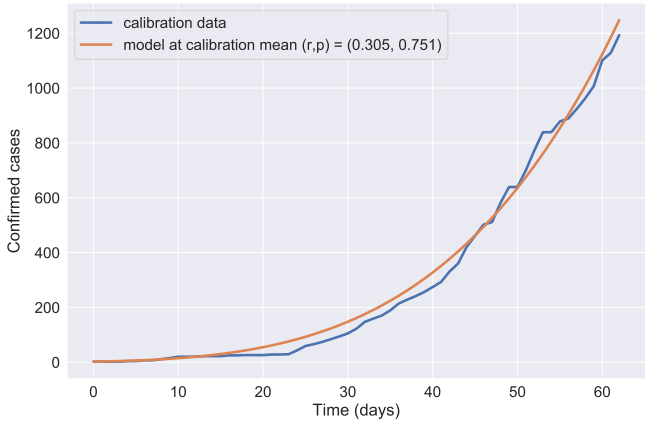


Figure 3: **Calibration result.**

4. Validation step

4.1. Gaussian approximation of the calibration posterior samples

We approximate the calibration posterior by Gaussian approximation of the calibration samples. The Initial value of parameter was taken as mean of the calibration samples. The acceptance rate in the Metropolis-Hastings was 0.035%. The mean of the sample is $\theta^v_{mean} = (r^v_{mean}, p^v_{mean}) = (0.132, 0.944)$. The model output using the mean of samples as parameter is shown in Fig. 6. The model output using the calibrated and the validated mean at two portions of time is shown in Fig. 7.

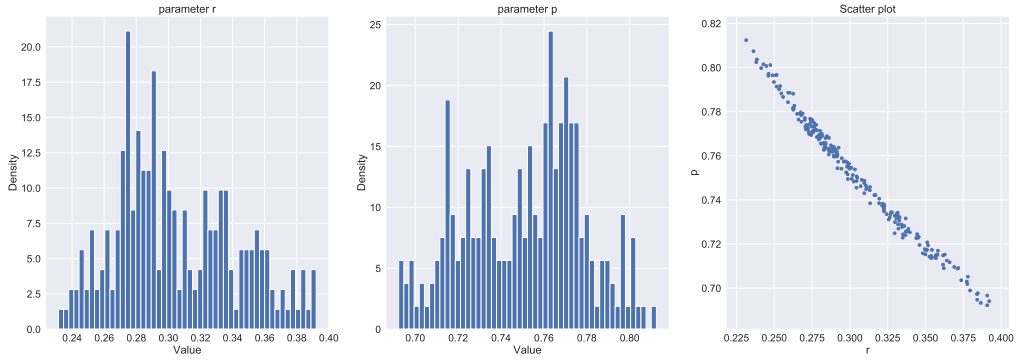


Figure 4: **Histogram and scatter plot of calibration posterior samples.**

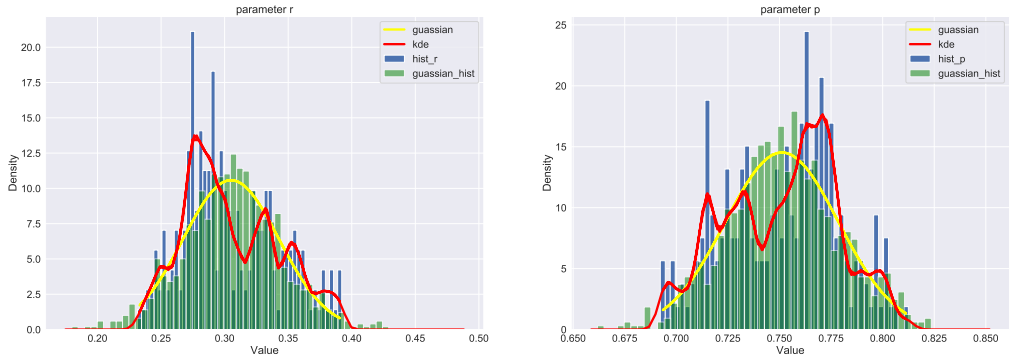


Figure 5: **Histogram, Gaussian approximation, and KDE approximation of posterior $\pi^c(\theta|Y^c)$.**

4.1.1. Validation error

The validation error e^v, see definition (18), is found to be 0.00505. This is below the $\gamma_{tol} = 0.01$ so we declare the model as "Not invalid".

4.2. KDE approximation of the calibration posterior samples

We approximate the calibration posterior by the KDE approximation of the calibration samples. The Initial value of parameter was taken as mean of the calibration samples. The acceptance rate in the Metropolis-Hastings was 0.031%. The mean of the sample is $\theta^v_{mean} = (r^v_{mean}, p^v_{mean}) = (0.132, 0.946)$. Model output using the mean of samples as parameter is shown in Fig. 8. The Model output using the calibrated and the validated mean at two portions of time is shown in Fig. 9.

4.2.1. Validation error

The validation error e^v, see definition (18), is found to be 0.00469. This is below the $\gamma_{tol} = 0.01$ so we declare the model as "Not invalid".

4.3. Effect of approximation of posterior samples

From the results, we see that there is no significant effect of changing Gaussian approximation with more accurate KDE approximation of calibration posterior samples. The error e^v in the case of Gaussian is 0.00505 and in the case of KDE is 0.00469.

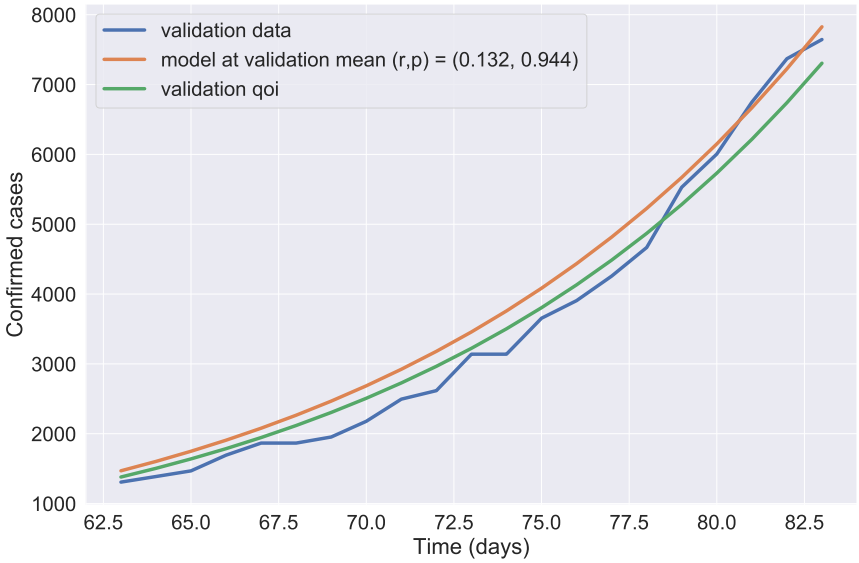


Figure 6: **Validation result with the Gaussian approximation of the calibration MCMC samples. The model output at mean of the validation MCMC samples is shown along with the mean of QoI distribution (see formula (17)). The mean of QoI is closer to the data which suggests that QoI should be computed using (17) taking into account the MCMC samples.**

5. Prediction and uncertainty in the prediction

5.1. Validation posterior obtained through Gaussian approximation of calibration posterior samples

Model prediction of QoI (number of cases at day 100) is $Q = 27858$ with the uncertainty 6026 i.e. 22%.

5.2. Validation posterior obtained through KDE approximation of calibration posterior samples

Model prediction of QoI (number of cases at day 100) is $Q = 29155$ with the uncertainty 6745 i.e. 23%.

5.3. QoI distribution

In Fig. 10, we show QoI distribution. In Fig. 11, we show the prediction by calibrated-validated model in the interval $[85, 100]$.

6. Conclusion

In this work, we have presented the Bayesian approach to the model fitting using the present day relevant COVID-19 data. This exercise shows various facets of the method and highlights the practical issues encountered during the numerical implementation. In our experience, the standard deviation in the noise affects the number of accepted samples. Higher the std deviation, the higher the accepted samples. This seems reasonable as higher σ weakens the likelihood function. This exercise also shows that the Gaussian approximation of the posterior samples performs as good as the KDE approximation. It will be interesting to consider a more complex model which incorporates multiple features such as spread of COVID-19 age wise, region-wise, and also multiscale models (in time). The codes used in this work will be freely available at Github site [2].

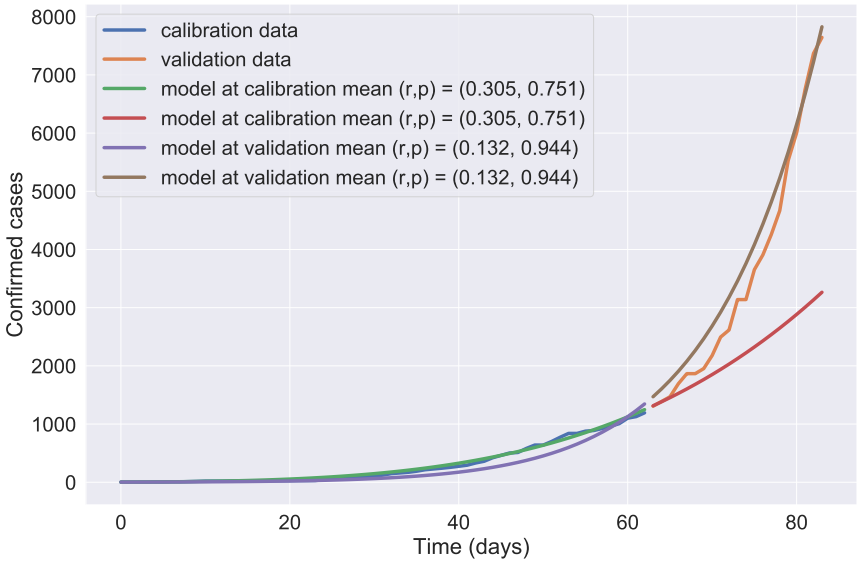


Figure 7: **Validation results with the Gaussian approximation of the calibration MCMC samples, and the calibration results.**

7. Acknowledgments

We would like to thank the AEOLUS group for supporting this work. We would also like to thank J. Moukarzel for presenting easy to read tutorial showing practical aspects of the Bayesian statistics and MCMC implementation, see his publicly available notebook [3].

References

[1] Hastings, W. K., 1970. Monte carlo sampling methods using markov chains and their applications.
[2] Jha, P. K., 2020. COVID 19 data. https://github.com/prashjha/StudyCovid19.
[3] Moukarzel, J., 2018. From scratch: Bayesian inference, Markov chain Monte Carlo and Metropolis Hastings, in python. https://github.com/Joseph94m/MCMC/blob/master/MCMC.ipynb.
[4] Oden, J., 2017. Foundations of predictive computational sciences. ICES Reports.
[5] Viboud, C., Simonsen, L., Chowell, G., 2016. A generalized-growth model to characterize the early ascending phase of infectious disease outbreaks. Epidemics 15, 27–37.

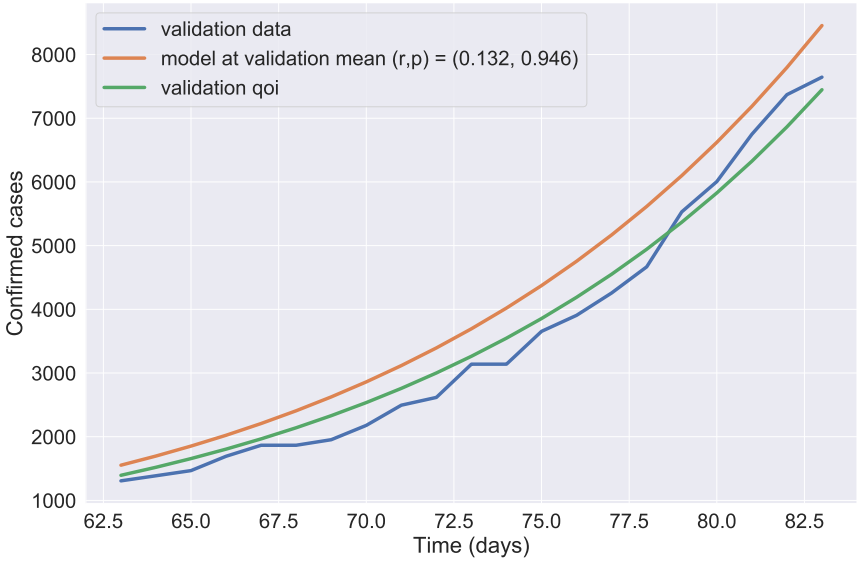


Figure 8: **Validation result with the KDE approximation of the calibration MCMC samples. The model output at mean of the validation MCMC samples is shown along with the mean of QoI distribution (see formula (17)).**

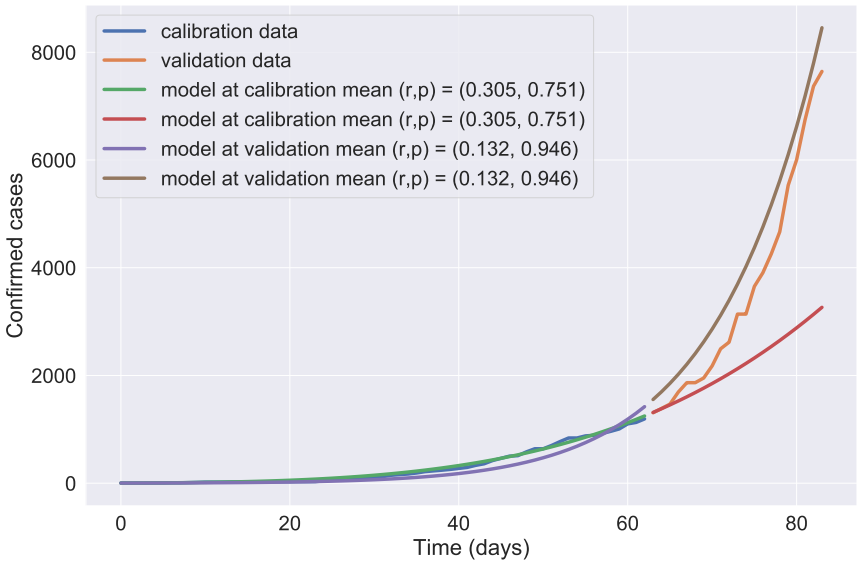


Figure 9: **Validation results with the KDE approximation of the calibration MCMC samples, and the calibration results.**

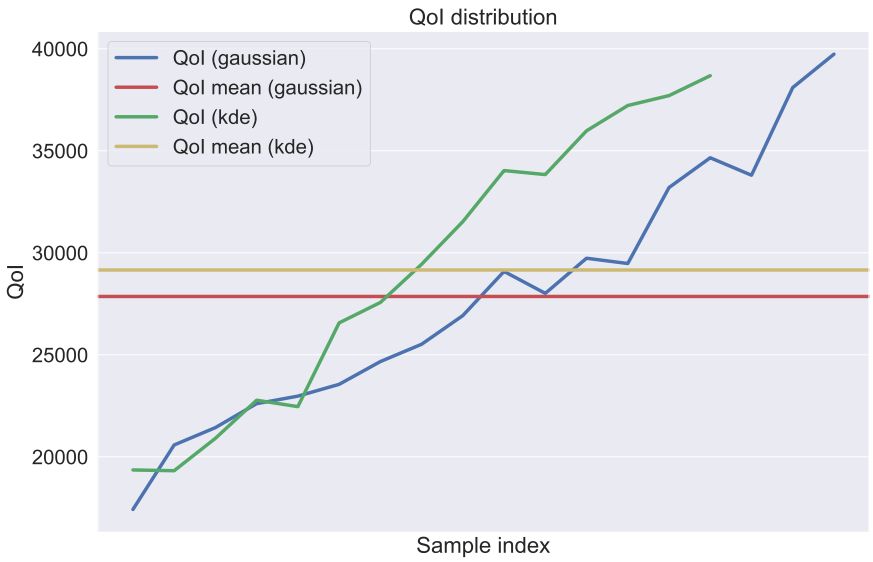


Figure 10: **QoI distribution along with the mean.**

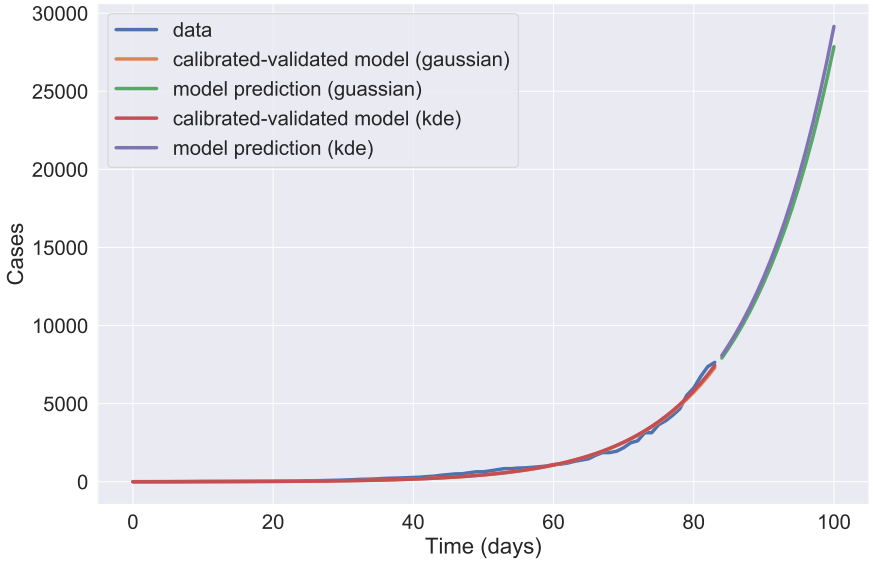


Figure 11: **Model prediction from day** 85 **till** 100**.**

Chapter 2

Bayesian Calibration, Validation, and Prediction with Application to Infectious Disease Models and COVID-19 Data

CSE 397 - Foundations of Predictive Computational Science

Lianghao Cao

April 28, 2020

For purpose of demonstration, consider the time evolution of the number of the novel Coronavirus infected cases in Japan, with data shown in the plot below.

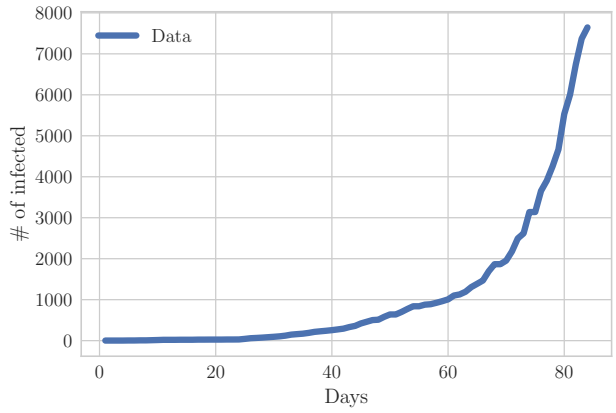


Figure 1: The time evolution of the recorded number of infected cases in Japan.

We consider an exponential growth model for the evolution of the infected cases:

$$C(t) = (\frac{r}{m}t + C_0^{1/m})^m \tag{1}$$

where $m = 1/(1-p)$ and $C_0 = C(0) = 1$ is the given initial condition. The parameter r and p are all positive number. Specifically, $p \in [0, 1]$.

1 The Calibration Step

With the given data $\boldsymbol{d} \in \mathbb{R}^{60}$ at $t = 1, 2, \ldots, 60$, the parameters $\boldsymbol{\theta} = [r, p]$ is calibrated via the Bayesian approach. The prior density for the parameter is given by a uniform density

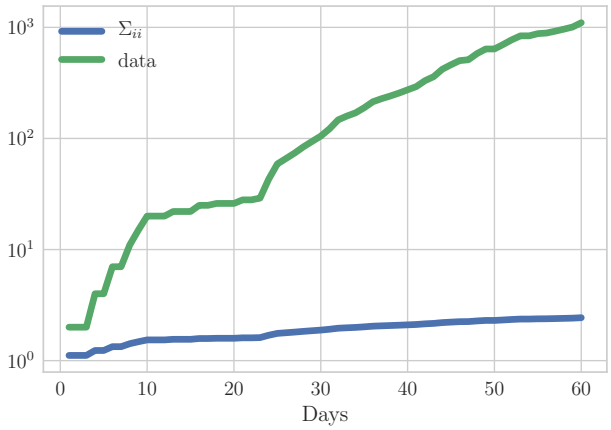


Figure 2: The diagonal entries of the covariance matrix of the Gaussian noise that follows (2) and data.

on the unit square $[0,1]^2$. The noise model on the data is naively set to be $\mathcal{N}(\mathbf{0}, \mathbf{\Sigma}^2)$[1], where $\mathbf{\Sigma} \in \mathbb{R}^{60\times 60}$ is a diagonal matrix with entries:

$$\Sigma_{ii} = 0.08 \ln(d_i) + 1 \ , \quad i = 1, 2, \ldots, 60 \ . \tag{2}$$

The diagonal entries of $\mathbf{\Sigma}^2$ is plotted with the provided data in Figure 2. This particular noise model partly reflects the assumption that the number of infected cases is noisier when it is large.

The log-likelihood function of a given parameter $\boldsymbol{\theta}$ is formulated via l^2 norm, scaled by the covariance of the noise model:

$$\sum_{i=1}^{60} \frac{1}{2\Sigma_{ii}^2} |C(\boldsymbol{\theta}, t_i) - d_i|^2, \quad i = 1, 2, \ldots, 60 \ . \tag{3}$$

The sampling method employed to obtain the equilibrium distribution of the posterior density is the MCMC with Metropolis-Hasting proposals, where the covariance matrix of the proposal Gaussian kernel has diagonal of 0.04.

The result of the calibration is shown in Figure 3 and 4 below[2]. The posterior density of the calibration step reveals that the parameters are highly correlated for the given data. The posterior density highly resembles the probability density of a multivariate normal distribution, with an estimated mean $\boldsymbol{\theta}_m \approx [0.2596, 0.7875]$.

[1]Ideally, the noise on the data should be strictly positive, meaning that the data could only underestimate the number of infected cases. For the purpose of demonstration and simplicity, we use the additive Gaussian noise, which is by no mean truthful.

[2]The full numerical implementation could be find at https://github.com/lcao11/bayes_covid

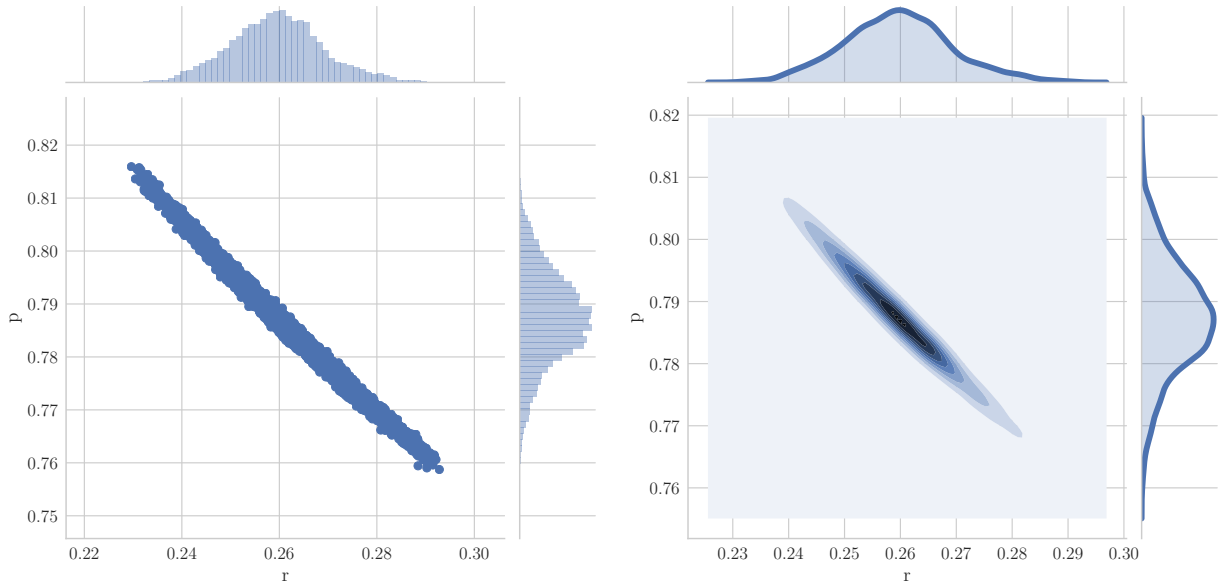


Figure 3: The posterior samples of the calibration step and their Gaussian kernel density estimation. In total samples of $100,000$, 37576 samples are accepted by the MCMC algorithm. The latter half of the chain is used to represent the calibration posterior above.

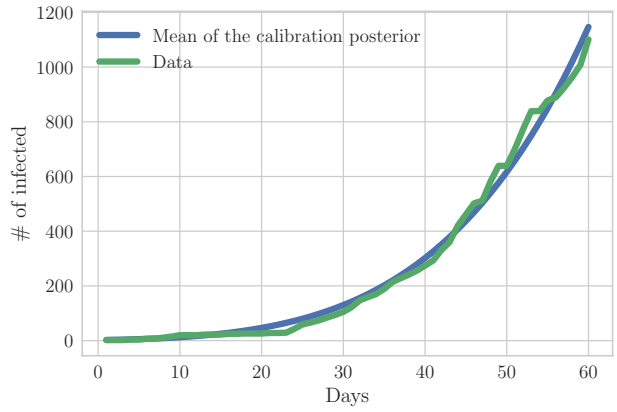


Figure 4: The forward solution corresponds to the mean of the calibration posterior compared to the data

2 The Validation Step

Using the additional data at $t = 61, 62, \ldots, 84$, a validation step via Bayesian approach is preformed. The prior density for the parameter is taken as the Gaussian kernel density estimation of the calibration posterior[3]. The same noise model, likelihood formulation, and sampling method as mentioned above are used.

The result for the validation step is shown in Figure 5, and 6. The validation posterior

[3]The Gaussian kernel density estimation is computed with `scipy.stats.gaussian_kde`

again resembles the probability density of a multivariate normal distribution, with estimated mean $\boldsymbol{\theta}_m = [0.2250, 0.8227]$.

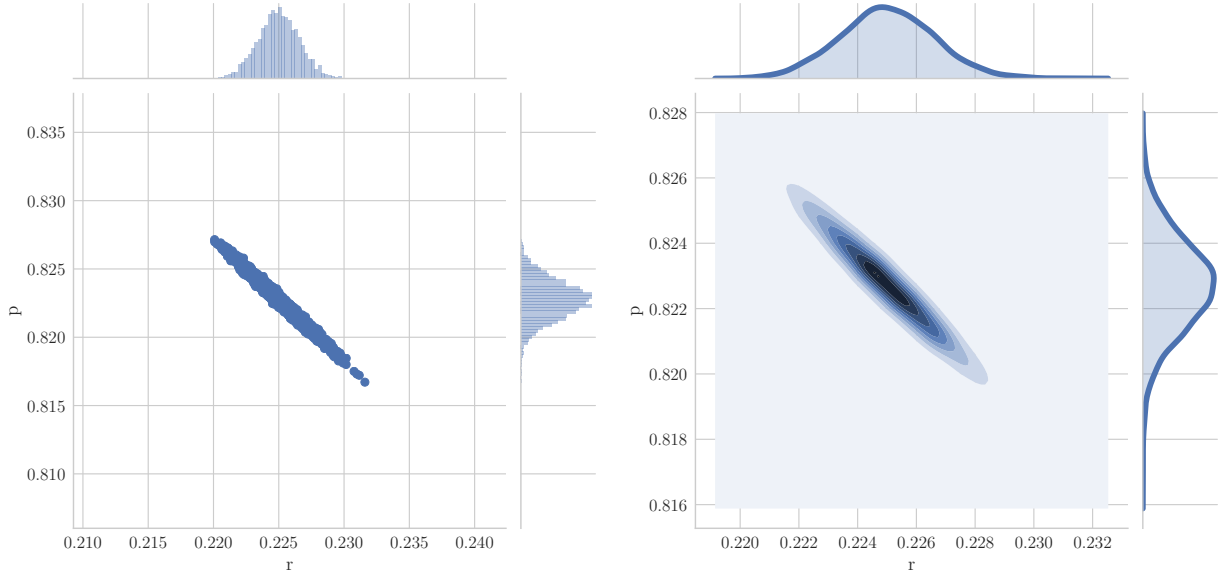


Figure 5: The posterior samples of the validation step and their Gaussian kernel density estimation. In total samples of 100,000, 8843 samples are accepted by the MCMC algorithm. The latter half of the chain is used to represent the validation posterior above.

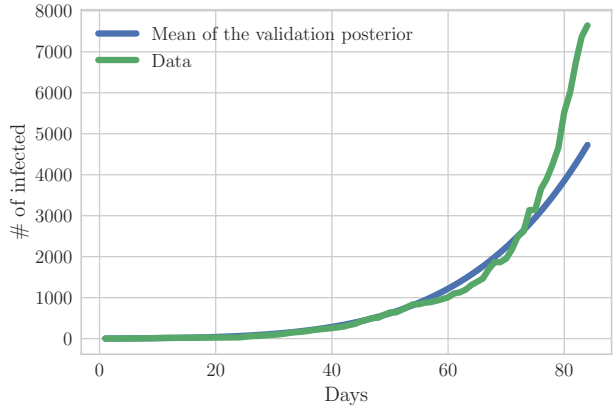


Figure 6: The forward solution corresponds to the mean of the validation posterior.

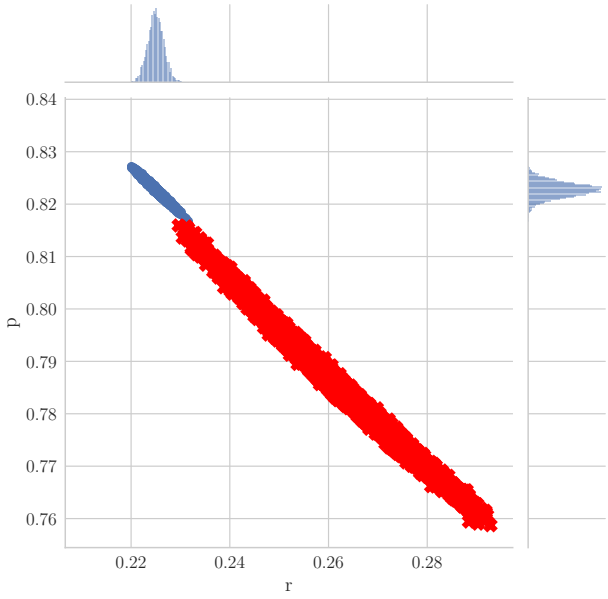


Figure 7: A comparison between the calibration posterior (red samples) and the validation posterior (blue samples).

With the collected validation posterior samples, the expectation value $\mathbb{E}_v[\cdot]$ and variance $Var_v[\cdot]$ of the total number of infected cases at $t = 84$ are calculated to be

$$\mathbb{E}_v[C(\cdot, 84)] \approx 4000 \,, \quad Var_v[C(\cdot, 84)] \approx 314.5 \,, \tag{4}$$

while the data has the total number of infected cases at 7645. The error in the quantity of interest is given by

$$\frac{|\mathbb{E}_v[C(\cdot, 84)] - d_{84}|}{d_{84}} \approx 0.4768 \,. \tag{5}$$

The large discrepancy between data and the expectation value of the validation posterior on the total number of infected cases at $t = 84$ indicates that we have an invalid forward model or (and) an invalid noise model.

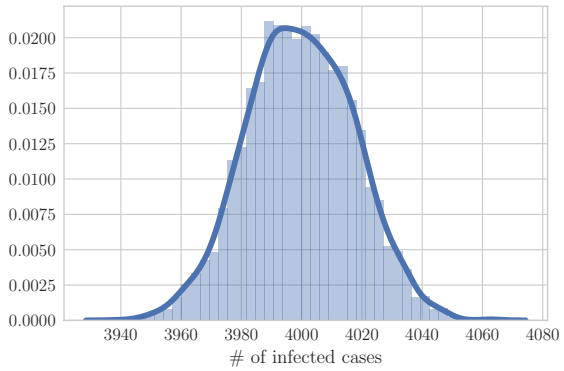


Figure 8: The Gaussian kernel density estimation of the distribution of the quantity of interest $C(84)$, the number of infected cases at day 84.

3 The Prediction Step

Despite our previous conclusion that the model is invalid, we use the validation posterior samples to predict the expectation value of the total number of infected cases at $t = 100$, and it turns out to be approximately 8657, with variance of 3239.

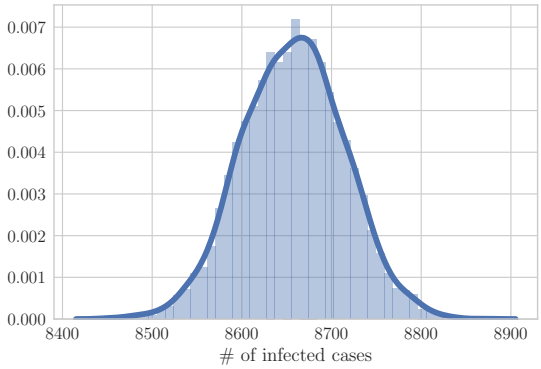


Figure 9: The Gaussian kernel density estimation of the distribution of the quantity of interest $C(100)$, the number of infected cases at day 100.

Chapter 3

CSE 397 Foundations of Predictive Computational Science

Taemin Heo (th29685)

COVID-19 Bayesian-statistical inverse calculation assignment

Model 1

$$C(t) = \left(\frac{r}{m}t + (C_0)^{\frac{1}{m}}\right)^m$$

where $m = \frac{1}{1-p}$ and $C_0 = C(0)$.

```
def model_1(t, r, p):
    m = 1/(1-p)
    return (r*t/m+C0**(1/m))**m
```

Data

We consider COVID 19 data at discrete times $\bar{\boldsymbol{t}} = (t_1 = 0, t_2 = 1, \dots, t_N = 83)$, where $N = 84$, and corresponding total confirmed cases $\boldsymbol{Y}(\bar{\boldsymbol{t}}) = (Y_1, Y_2, \dots, Y_N)$.

We partition the data into two sets, $\boldsymbol{y}_c$ for calibration and $\boldsymbol{y}_v$ for validation,

$$\boldsymbol{y}_c = \left(\bar{t}_c, Y(\bar{t}_c)\right)$$

where $\bar{t}_c = \left((t_1 = 0, t_2 = 1, \dots, t_{50} = 49)\right)$,

$$\boldsymbol{y}_v = \left(\bar{t}_v, Y(\bar{t}_v)\right)$$

where $\bar{t}_c = \left((t_{51} = 50, t_{52} = 51, \dots, t_N = 83)\right)$.

QoI

$$C(100)$$

Parameter

$$\boldsymbol{\theta_{m1}} = (r, p)$$

Prior

$$\pi_{prior}(r) \sim Unif\left([0, \infty)\right)$$

$$\pi_{prior}(p) \sim Unif([0,1])$$

The statistical independence between r, p is assumed.

$$\therefore \pi_{prior}(\boldsymbol{\theta_{m1}}) = \pi_{prior}(r)\pi_{prior}(p)$$

```
def prior_1(x):
    #x[0] = r, x[1]=p (new or current)
    if(x[0] <= 1.e-5):
        return 1.e-8
    elif (x[1] <= 1.e-5 or x[1] >= 1. - 1.e-5):
        return 1.e-8
    else:
        return 1.
```

Likelihood

$$\pi_{like} \propto \exp\left[-\frac{\|\boldsymbol{Y} - C(\boldsymbol{\theta_{m1}})\|^2}{2\sigma^2}\right]$$

where $C(\boldsymbol{\theta_{m1}})$ is the model output when parameter is $\boldsymbol{\theta_{m1}}$ and $\|\boldsymbol{Y} - C(\boldsymbol{\theta_{m1}})\|^2$ is given by

$$\|\boldsymbol{Y} - \boldsymbol{C}(\boldsymbol{\theta_{m1}})\|^2 = \sum_{i=1}^{n} |Y_i - C(t_i; \boldsymbol{\theta_{m1}})|^2.$$

Then, the logarithmic likelihood function is

$$\log(\pi_{like}) \propto -\frac{1}{2\sigma^2} \sum_{i=1}^{n} |Y_i - C(t_i; \boldsymbol{\theta_{m1}})|^2.$$

Here, the standard deviation of residual will be assumed as 500. In context, it means that the confirmed cases number has noise of zero mean and standard deviation of 100 people.

```
def log_like_1(x,data):
    #x[0]=r, x[1]=p (new or current)
    #data[0]=t, data[1]=C_obs(t)
    C = model_1(data[0],x[0],x[1])
    return (-1. / (2. * noise_sig * noise_sig)) * np.dot(data[1] - C,data[1] - C)
```

Calibration

The posterior updated by S_c via Bayes' rule,

$$\pi_{post}(\boldsymbol{\theta_{m1}}|\boldsymbol{y}_c, S_c) = \frac{\pi_{like}(\boldsymbol{y}_c|\boldsymbol{\theta_{m1}}, S_c)\pi_{prior}(\boldsymbol{\theta_{m1}})}{\pi(\boldsymbol{y}_c|S_c)}.$$

The calibration posterior is obtained by Metropolis-Hastings MCMC,

```
#Defines whether to accept or reject the new sample
def acceptance(x, x_new):
    if x_new>x:
        return True
    else:
        accept=np.random.uniform(0,1)
        # Since we did a log likelihood, we need to exponentiate in order to compare to
 the random number
        # less likely x_new are less likely to be accepted
        return (accept < (np.exp(x_new-x)))
```

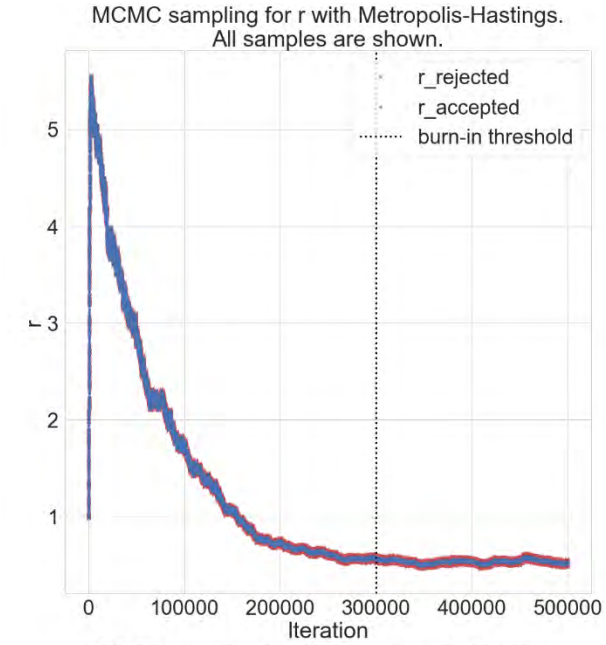
```
10.
11. def metropolis_hastings(likelihood_computer,prior, transition_model, param_init,iterati
    ons,data,acceptance_rule):
12.     # likelihood_computer(x,data): returns the likelihood that these parameters generat
    ed the data
13.     # transition_model(x): a function that draws a sample from a symmetric distribution
     and returns it
14.     # param_init: a starting sample
15.     # iterations: number of accepted to generated
16.     # data: the data that we wish to model
17.     # acceptance_rule(x,x_new): decides whether to accept or reject the new sample
18.     x = param_init
19.     accepted = []
20.     rejected = []
21.     for i in range(iterations):
22.         x_new =  transition_model(x)
23.         x_lik = likelihood_computer(x,data)
24.         x_new_lik = likelihood_computer(x_new,data)
25.         if (acceptance_rule(x_lik + np.log(prior(x)),x_new_lik+np.log(prior(x_new)))):

26.             x = x_new
27.             accepted.append(x_new)
28.         else:
29.             rejected.append(x_new)
30.
31.     return np.array(accepted), np.array(rejected)
```

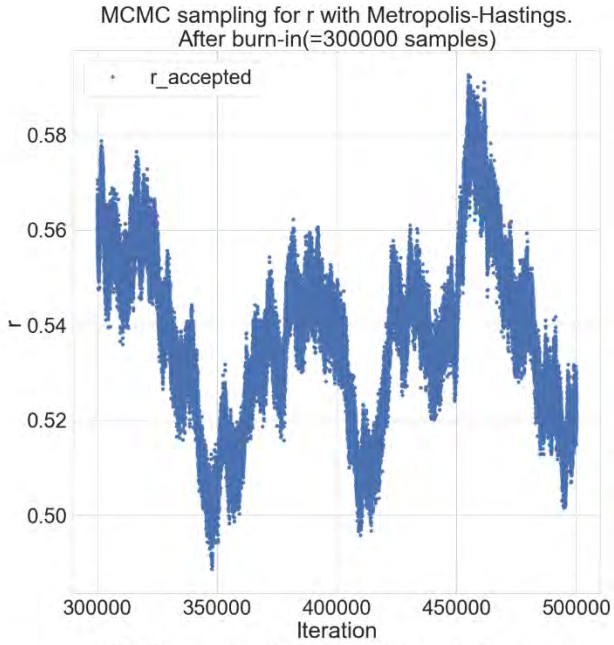
The transition model, standard deviation of the noise, and initial guess of the parameters are set as follows.

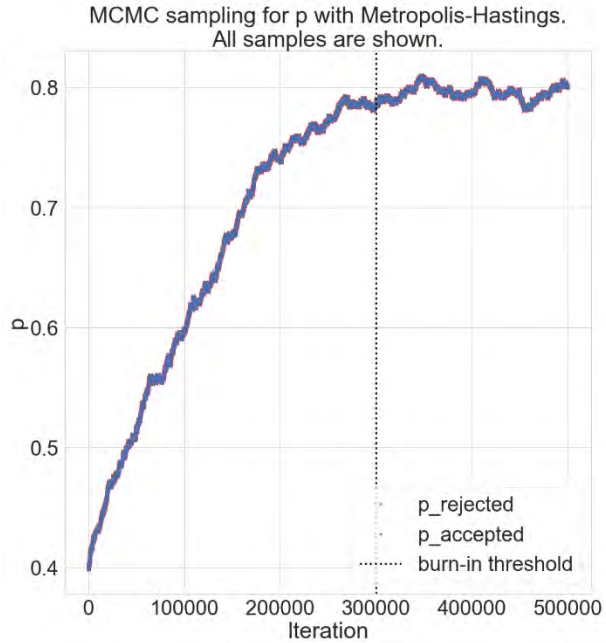
```
1. transition_model_m1c = lambda x: np.random.normal(x,[0.01,0.0001],(2,))
2. noise_sig = 500.
3. theta0_m1c = [1., 0.4]
4. accepted_m1c, rejected_m1c, itr_a_m1c, itr_r_m1c = metropolis_hastings(log_like_1,prior
   _1,transition_model_m1c,theta0_m1c,500000,data[:,:50],acceptance)
```

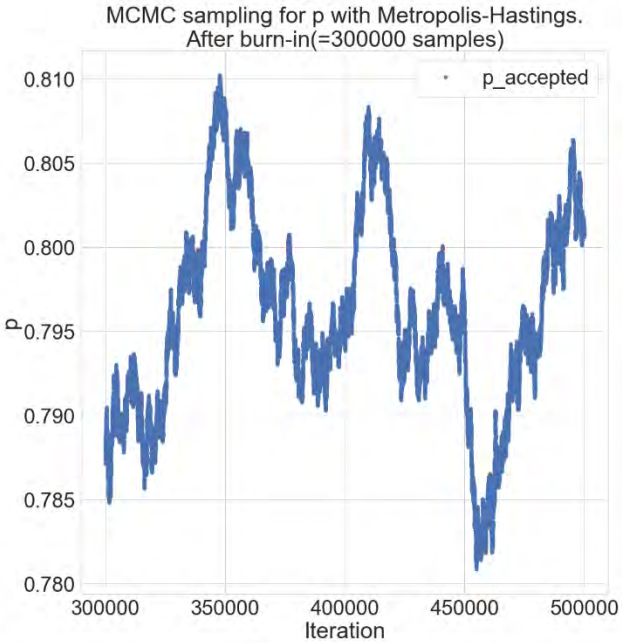
Calibration Result

MCMC convergence

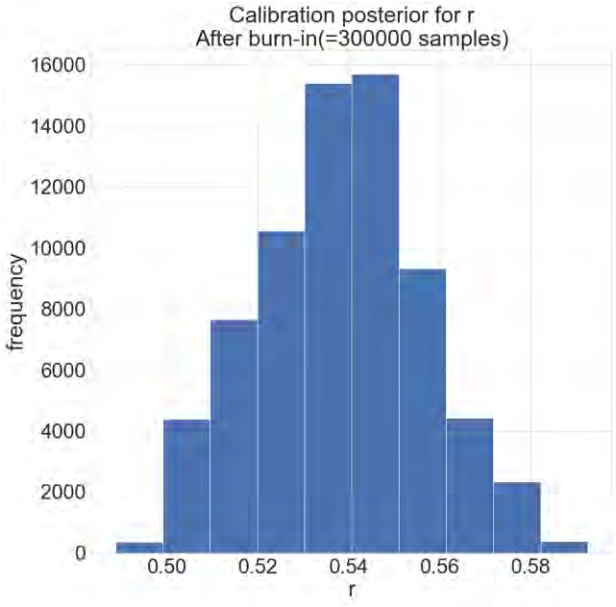


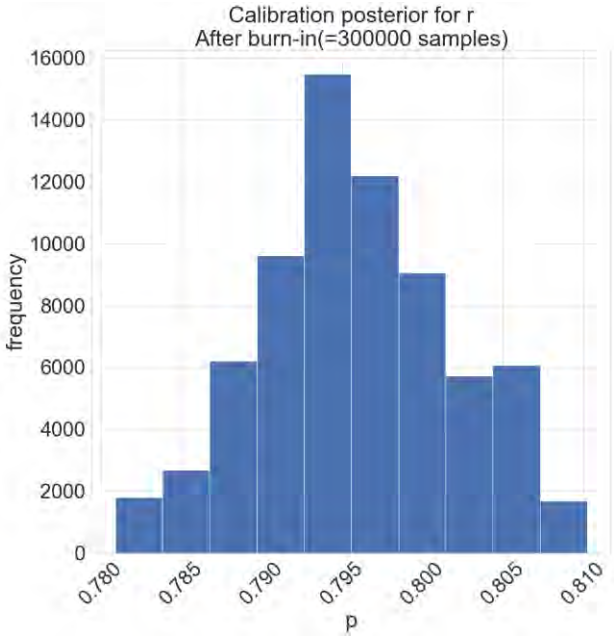




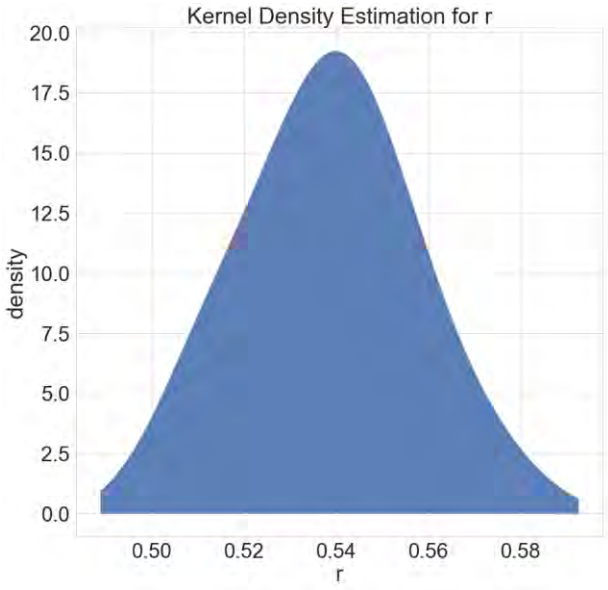


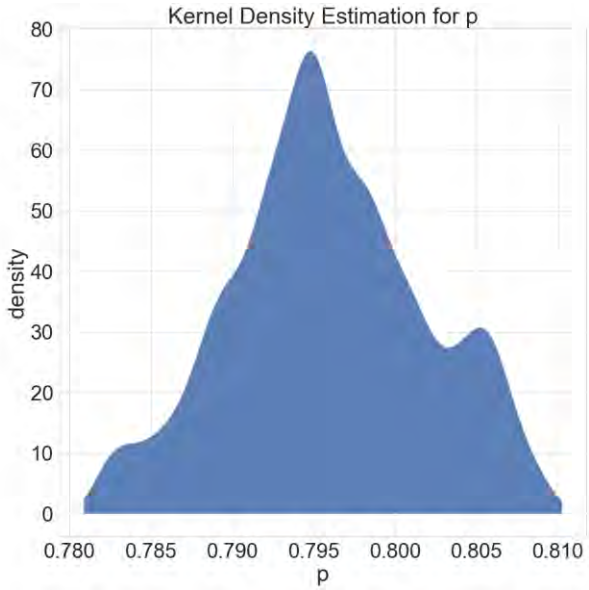
Calibration Posterior





Kernel Density Estimation

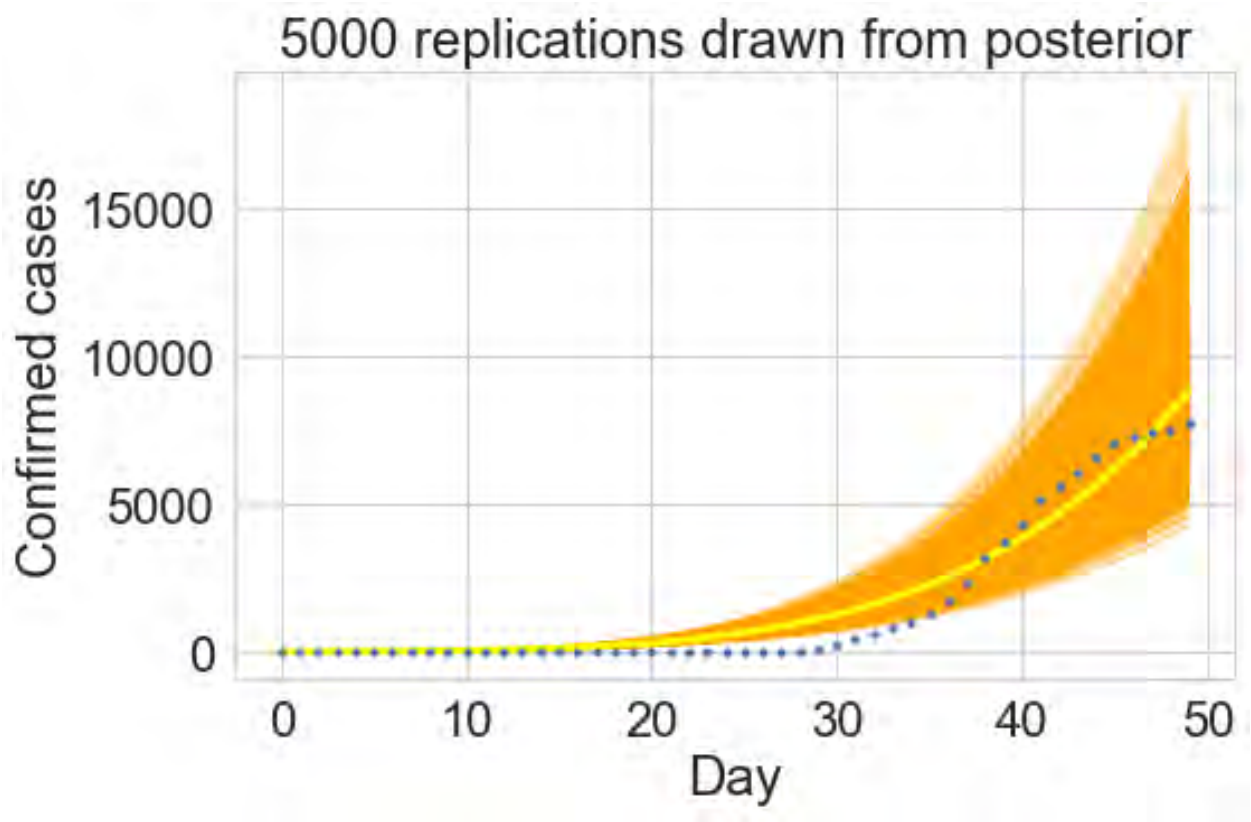




Gaussian kernel with bandwidth = 0.01 is used for r.

Gaussian kernel with bandwidth = 0.001 is used for p.

Calibration graphical consistency check



Validation

The posterior updated by S_v via Bayes' rule,

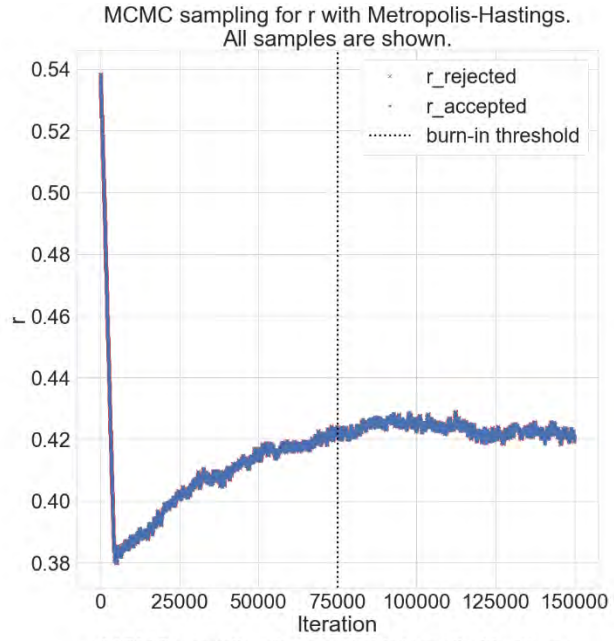
$$\pi_{post}(\boldsymbol{\theta_{m1}}|\boldsymbol{y}_v, \boldsymbol{y}_c, S_v, S_c) = \frac{\pi_{like}(\boldsymbol{y}_v|\boldsymbol{\theta_{m1}}, \boldsymbol{y}_c, S_c, S_v)\pi_{post}(\boldsymbol{\theta_{m1}}|\boldsymbol{y}_c, S_c)}{\pi(\boldsymbol{y}_v|\boldsymbol{y}_c, S_v)}.$$

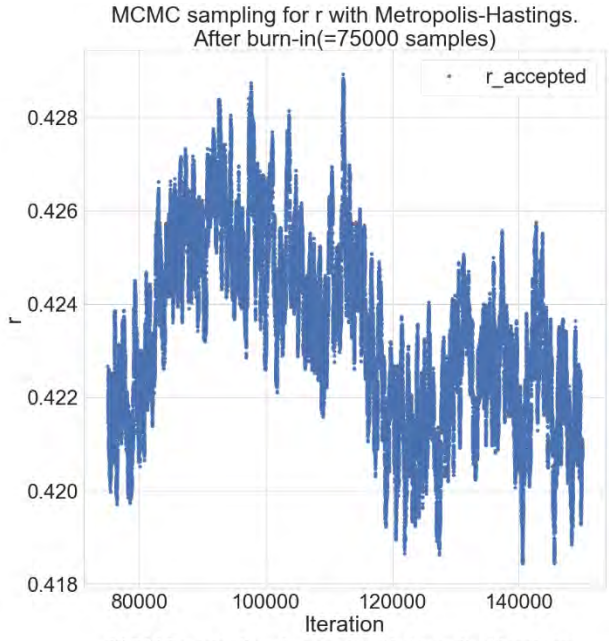
The validation predictive posterior is obtained by Metropolis-Hastings MCMC as we did for the calibration.

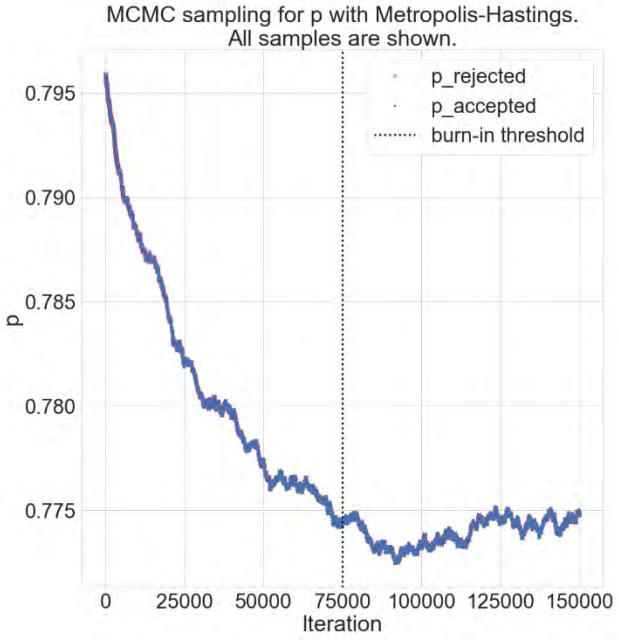
```
#calibration posterior
#prior distribution for validation
def prior_c_1(x):
    #x[0] = r, x[1]=p (new or current)
    logprob_r = kde_r.score_samples(x[0].reshape(1,-1))
    logprob_p = kde_p.score_samples(x[1].reshape(1,-1))
    return np.exp(logprob_r + logprob_p)

%%capture --no-display
transition_model_m1v = lambda x: np.random.normal(x,[0.0001,0.00001],(2,))
noise_sig = 500.
accepted_m1v, rejected_m1v, itr_a_m1v, itr_r_m1v = metropolis_hastings(log_like_1,prior
_c_1,transition_model_m1v,mu_m1c,150000,data[:,50:],acceptance)
```

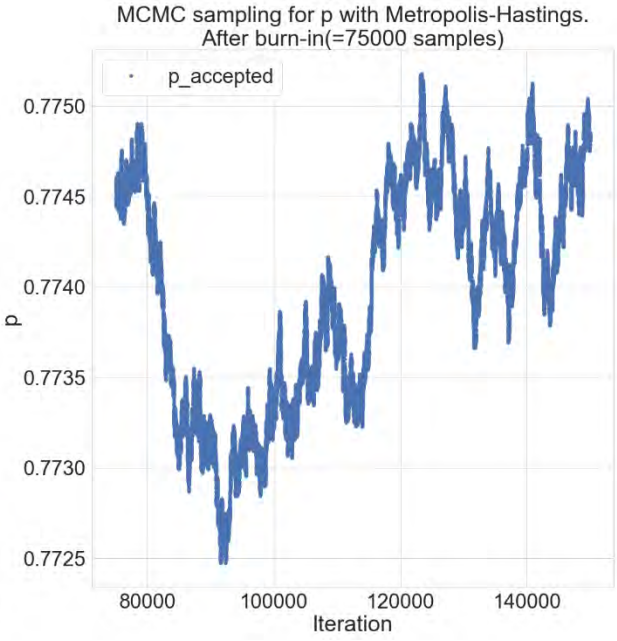
Validation Result

MCMC convergence

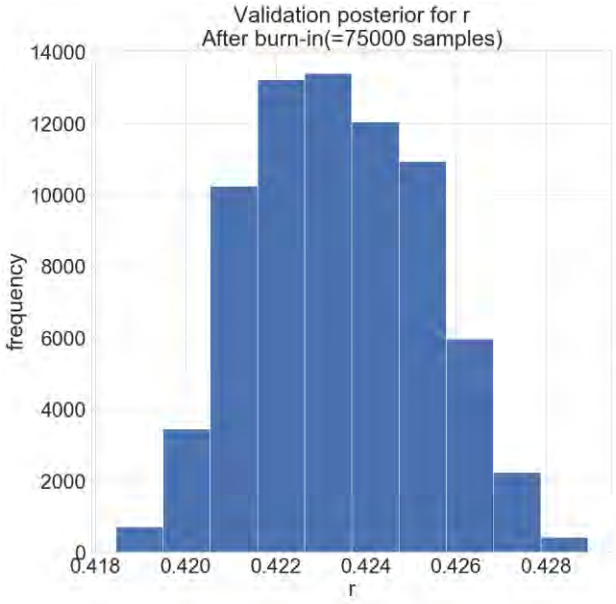


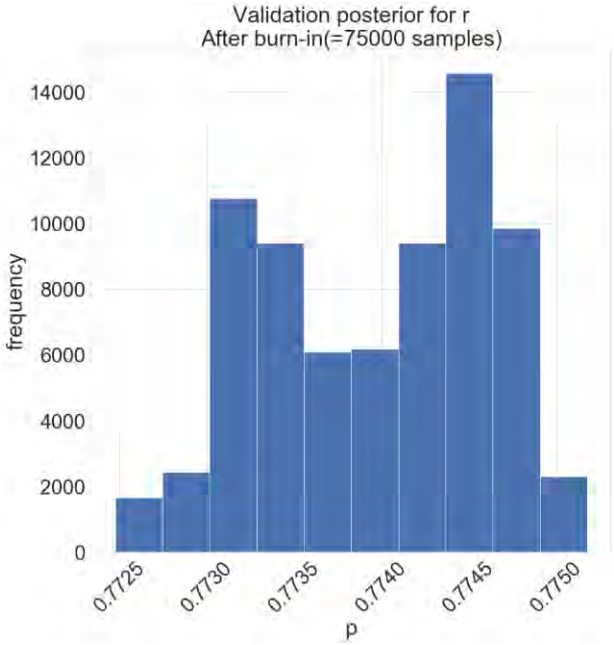




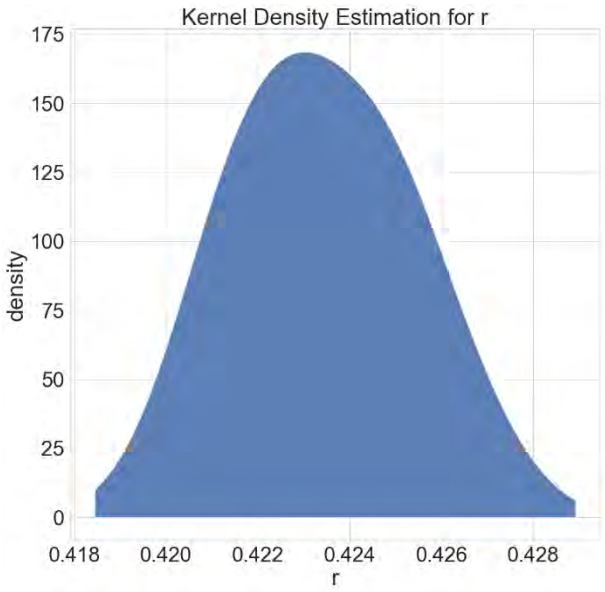


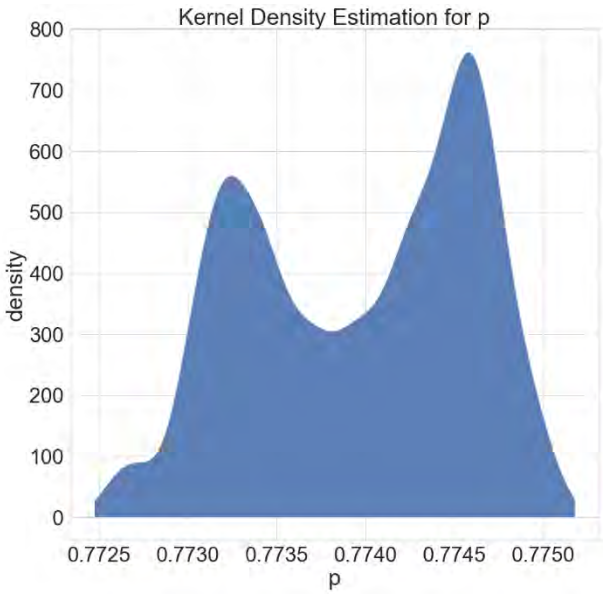
Validation Posterior





Kernel Density Estimation

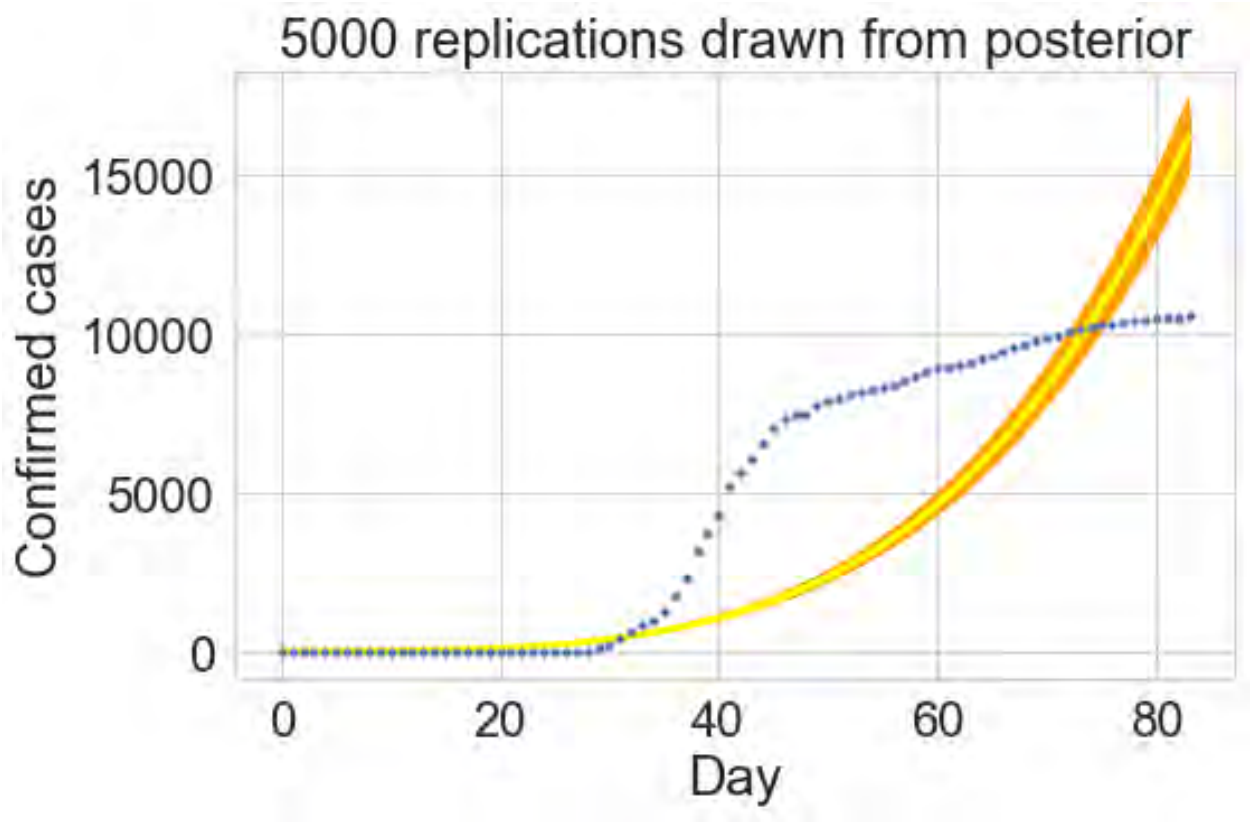




Gaussian kernel with bandwidth = 0.001 is used for r.

Gaussian kernel with bandwidth = 0.0001 is used for p.

Validation graphical accuracy check



Due to small variance of validation posterior predictive distribution, 5,000 replications drawn from the validation posterior predictive posterior has small band as shown above Figure.

However, as we can see from the Figure the validation predictions accuracy is poor. This is because the limitation of the model.

Here, the metric for the validation defined as below.

$$|C(83) - Y_{84}| < 0.1 * Y_{84}$$

This metric is devised from our quantity of interest, $C(100)$. Since our QoI is predicting the confirmed cases of far-future, the error at the most far day that we can validate is selected as our metric.

For the model 1, the metric $|C_{m1}(83) - Y_{84}| = 0.53 * Y_{84}$. Therefore, we conclude that the calibrated model 1 is invalid.

Model 3

$$C(t) = a \exp\left[b\left(1 - \frac{1}{1-\left(1-\frac{t}{T}\right)^{P}}\right)\right] + C_0$$

where a, b, p are model parameters. T is the bound on time beyond which this model will fail to work. This is the limitation of this model. We can take $T = 100$.

```
1.  def model_3(t, a, b, p):
2.      return a * np.exp(b*(1-1/(1-(1-t/100)**p)))+C0
```

Data

We consider COVID 19 data at discrete times $\bar{\boldsymbol{t}} = (t_1 = 0, t_2 = 1, \dots, t_N = 83)$, where $N = 84$, and corresponding total confirmed cases $\boldsymbol{Y}(\bar{\boldsymbol{t}}) = (Y_1, Y_2, \dots, Y_N)$.

We partition the data into two sets, $\boldsymbol{y}_c$ for calibration and $\boldsymbol{y}_v$ for validation,

$$\boldsymbol{y}_c = \left(\bar{t}_c, Y(\bar{t}_c)\right)$$

where $\bar{t}_c = \left((t_1 = 0, t_2 = 1, \dots, t_{50} = 49)\right)$,

$$\boldsymbol{y}_v = \left(\bar{t}_v, Y(\bar{t}_v)\right)$$

where $\bar{t}_c = \left((t_{51} = 50, t_{52} = 51, \dots, t_N = 83)\right)$.

QoI

$$C(100)$$

Parameter

$$\boldsymbol{\theta_{m3}} = (a, b, p)$$

Prior

$$\pi_{prior}(a) \sim Unif\left([0, \infty)\right)$$

$$\pi_{prior}(b) \sim Unif\left([0, \infty)\right)$$

$$\pi_{prior}(p) \sim Unif\left([1, \infty)\right)$$

The statistical independence between a, b, p is assumed.

$$\therefore \pi_{prior}(\boldsymbol{\theta_{m3}}) = \pi_{prior}(a)\pi_{prior}(b)\pi_{prior}(p)$$

```
1.  def prior_3(x):
2.      #x[0] = a, x[1]=b, x[2]=p (new or current)
3.      if(x[0] <= 1.e-5):
```

```
        return 1.e-8
    elif (x[1] <= 1.e-5):
        return 1.e-8
    elif (x[2] <= 1. + 1.e-5):
        return 1.e-8
    else:
        return 1.
```

Likelihood

$$\pi_{like} \propto \exp\left[-\frac{\|\boldsymbol{Y} - C(\boldsymbol{\theta_{m3}})\|^2}{2\sigma^2}\right]$$

where $C(\boldsymbol{\theta_{m3}})$ is the model output when parameter is $\boldsymbol{\theta_{m3}}$ and $\|\boldsymbol{Y} - C(\boldsymbol{\theta_{m3}})\|^2$ is given by

$$\|\boldsymbol{Y} - \boldsymbol{C}(\boldsymbol{\theta_{m3}})\|^2 = \sum_{i=1}^{n} |Y_i - C(t_i; \boldsymbol{\theta_{m3}})|^2.$$

Then, the logarithmic likelihood function is

$$\log(\pi_{like}) \propto -\frac{1}{2\sigma^2} \sum_{i=1}^{n} |Y_i - C(t_i; \boldsymbol{\theta_{m3}})|^2.$$

Here, the standard deviation of residual will be assumed same as 500.

```
def log_like_3(x,data):
    #x[0]=a, x[1]=b, x[2]=p (new or current)
    #data[0]=t, data[1]=C_obs(t)
    C = model_3(data[0],x[0],x[1],x[2])
    return (-1. / (2. * noise_sig * noise_sig)) * np.dot(data[1] - C,data[1] - C)
```

Calibration

The posterior updated by S_c via Bayes' rule,

$$\pi_{post}(\boldsymbol{\theta_{m3}}|\boldsymbol{y}_c, S_c) = \frac{\pi_{like}(\boldsymbol{y}_c|\boldsymbol{\theta_{m3}}, S_c)\pi_{prior}(\boldsymbol{\theta_{m3}})}{\pi(\boldsymbol{y}_c|S_c)}.$$

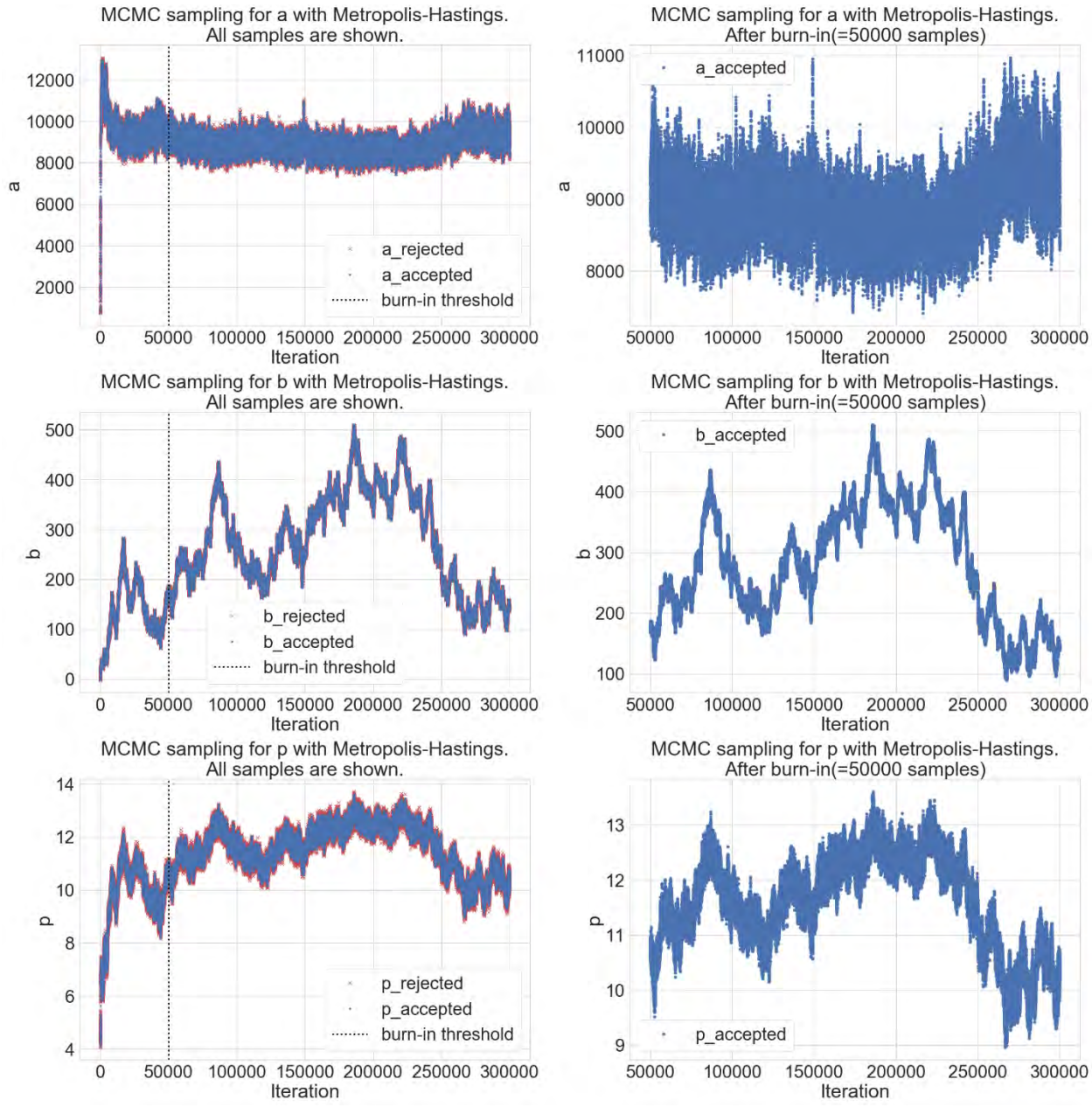
The calibration posterior is obtained by Metropolis-Hastings MCMC.

The transition model, standard deviation of the noise, and initial guess of the parameters are set as follows.

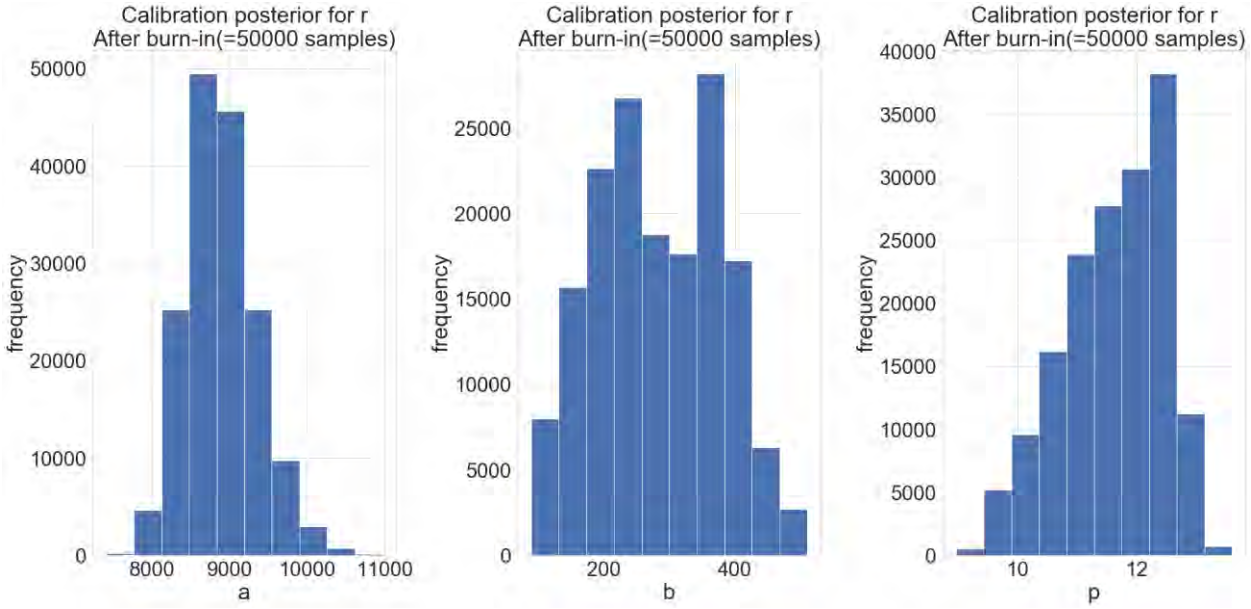
```
transition_model_m3c = lambda x: np.random.normal(x,[100.,1.,0.1],(3,))
noise_sig = 500.
theta0_m3c = [0.1*np.max(data[1,:50]), 0.4, 4.]
accepted_m3c, rejected_m3c, itr_a_m3c, itr_r_m3c = metropolis_hastings(log_like_3,prior
_3,transition_model_m3c,theta0_m3c,300000,data[:,:50],acceptance)
```

Calibration Result

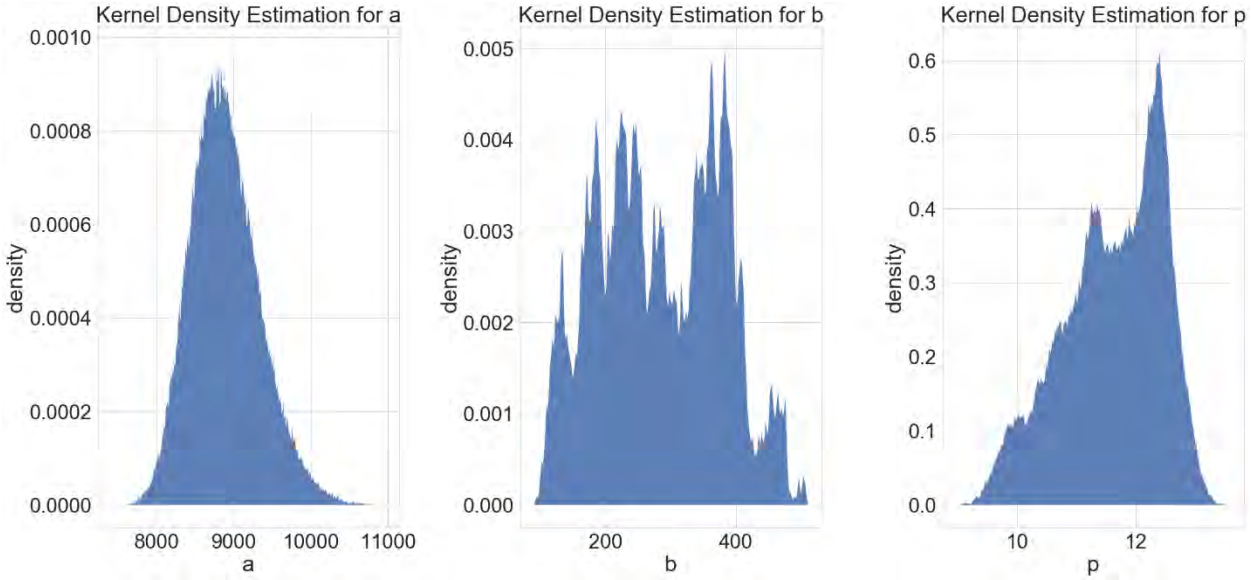
MCMC convergence



Calibration Posterior



Kernel Density Estimation

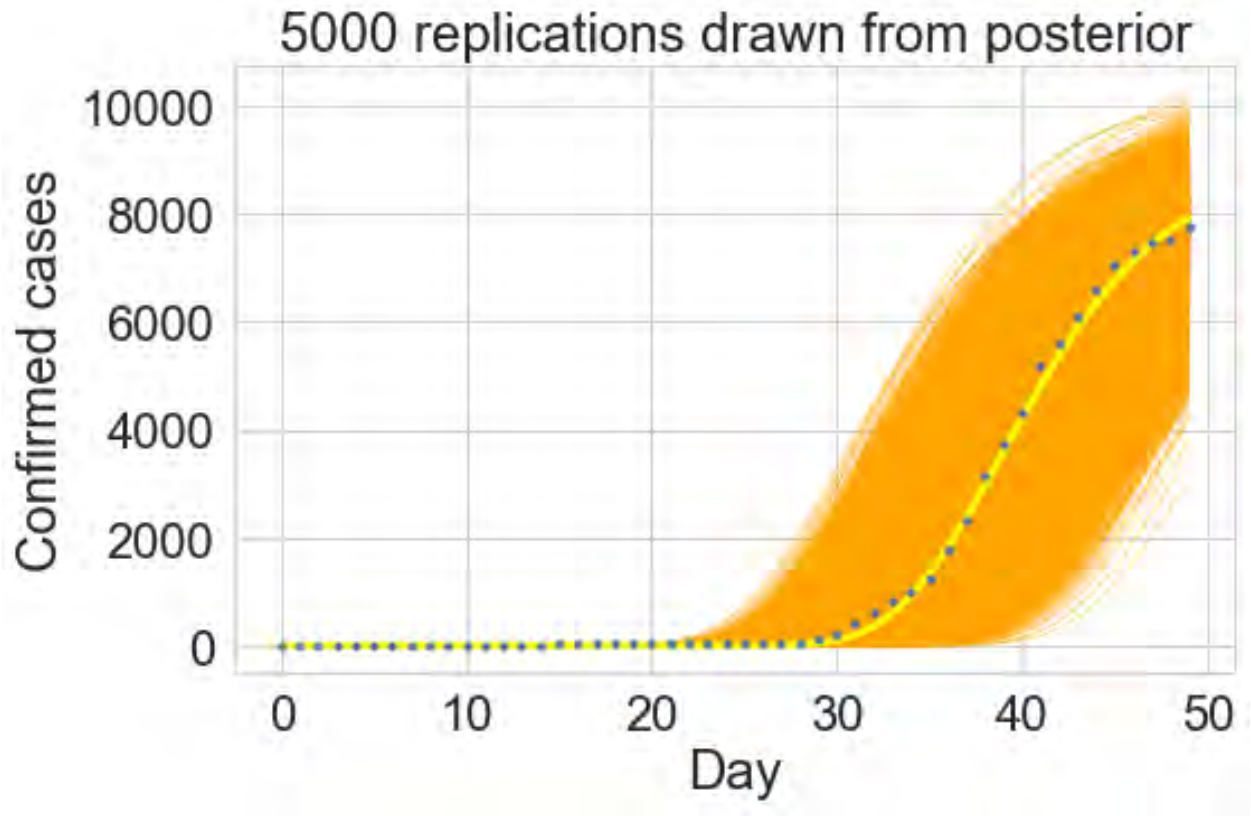


Gaussian kernel with bandwidth = 3.0 is used for a.

Gaussian kernel with bandwidth = 1.0 is used for b.

Gaussian kernel with bandwidth = 0.01 is used for p.

Graphical consistency check



Validation

The posterior updated by S_v via Bayes' rule,

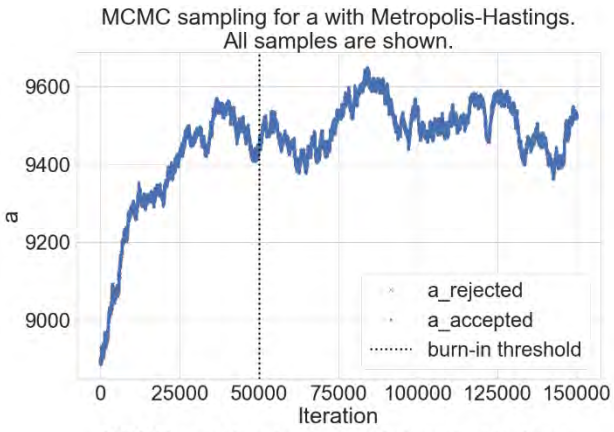
$$\pi_{post}(\boldsymbol{\theta}|\boldsymbol{y}_v, \boldsymbol{y}_c, S_v, S_c) = \frac{\pi_{like}(\boldsymbol{y}_v|\boldsymbol{\theta}, \boldsymbol{y}_c, S_c, S_v)\pi_{post}(\boldsymbol{\theta}|\boldsymbol{y}_c, S_c)}{\pi(\boldsymbol{y}_v|\boldsymbol{y}_c, S_v)}.$$

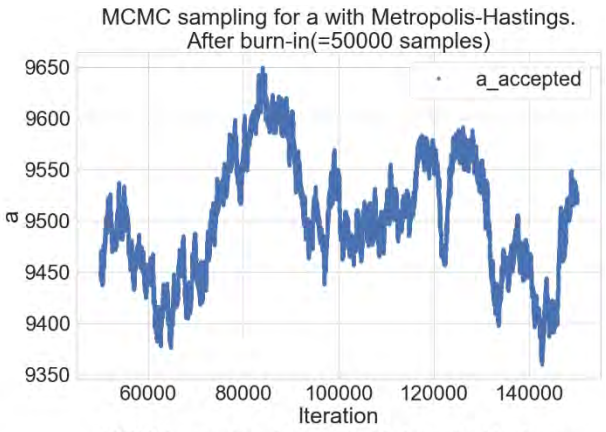
The validation predictive posterior is obtained by Metropolis-Hastings MCMC as we did for the calibration.

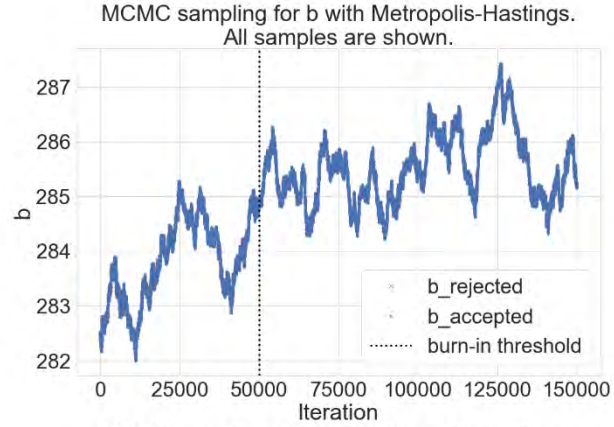
```
#calibration posterior
#prior distribution for validation
def prior_c_3(x):
    #x[0] = r, x[1]=p (new or current)
    logprob_a = kde_a.score_samples(x[0].reshape(1,-1))
    logprob_b = kde_b.score_samples(x[1].reshape(1,-1))
    logprob_p = kde_p.score_samples(x[2].reshape(1,-1))
    return np.exp(logprob_a + logprob_b + logprob_p)

%%capture --no-display
transition_model_m3v = lambda x: np.random.normal(x,[1.,0.01,0.001],(3,))
noise_sig = 500.
accepted_m3v, rejected_m3v, itr_a_m3v, itr_r_m3v = metropolis_hastings(log_like_3,prior
_c_3,transition_model_m3v,mu_m3c,150000,data[:,50:],acceptance)
```

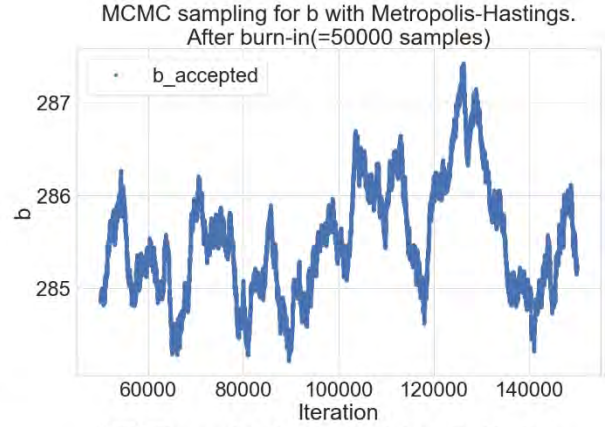
Validation Result

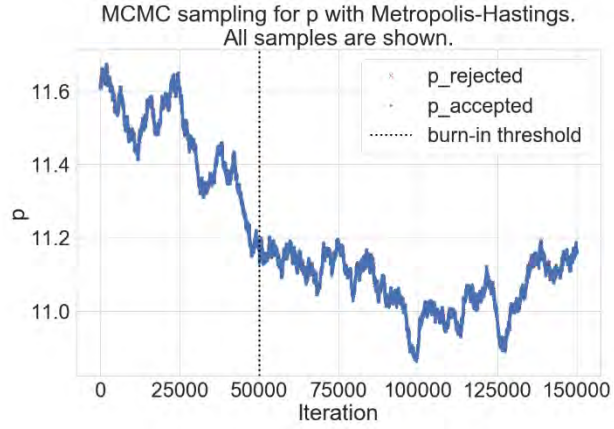
MCMC convergence

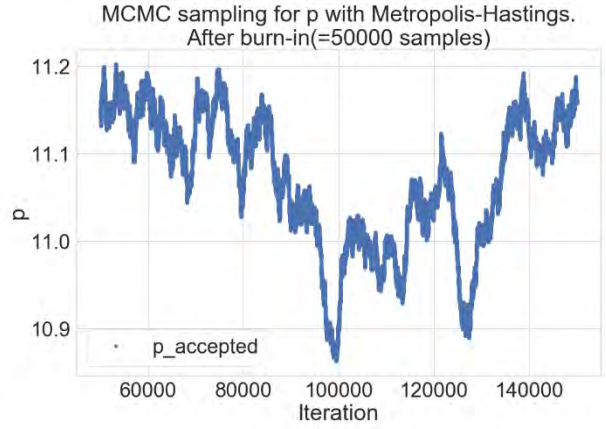




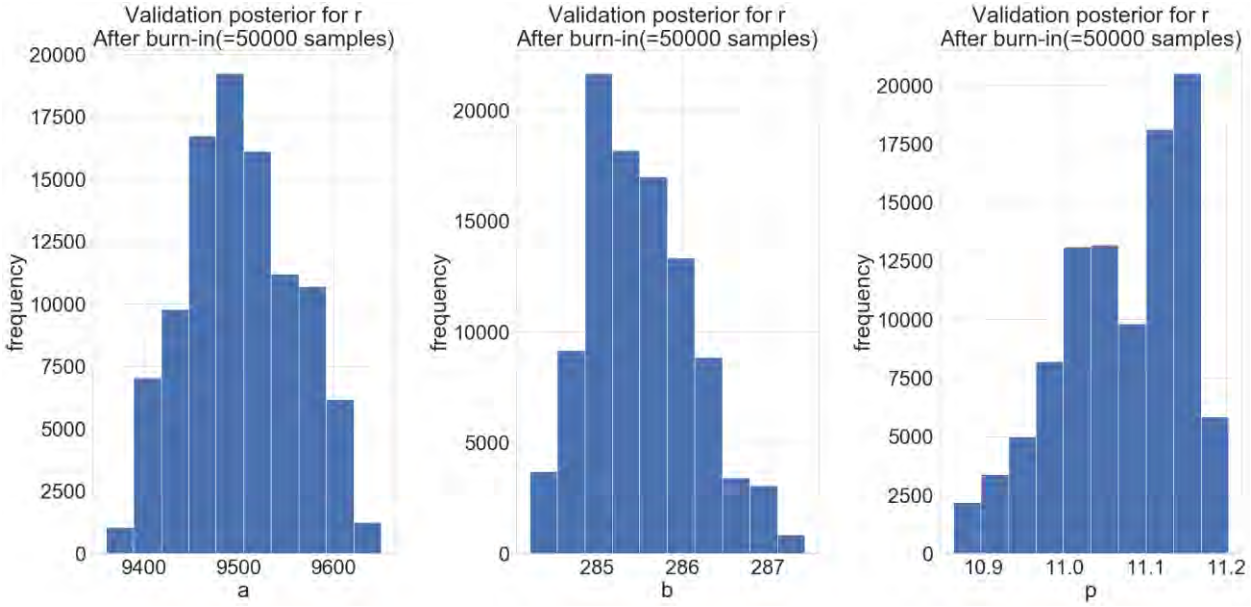




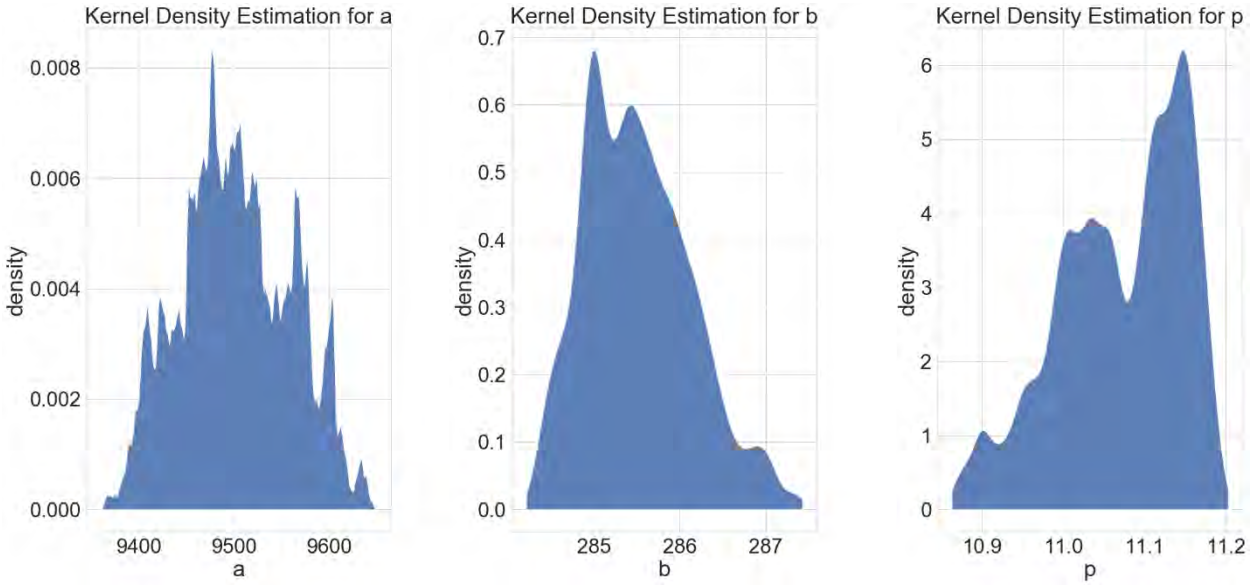




Validation Posterior



Kernel Density Estimation

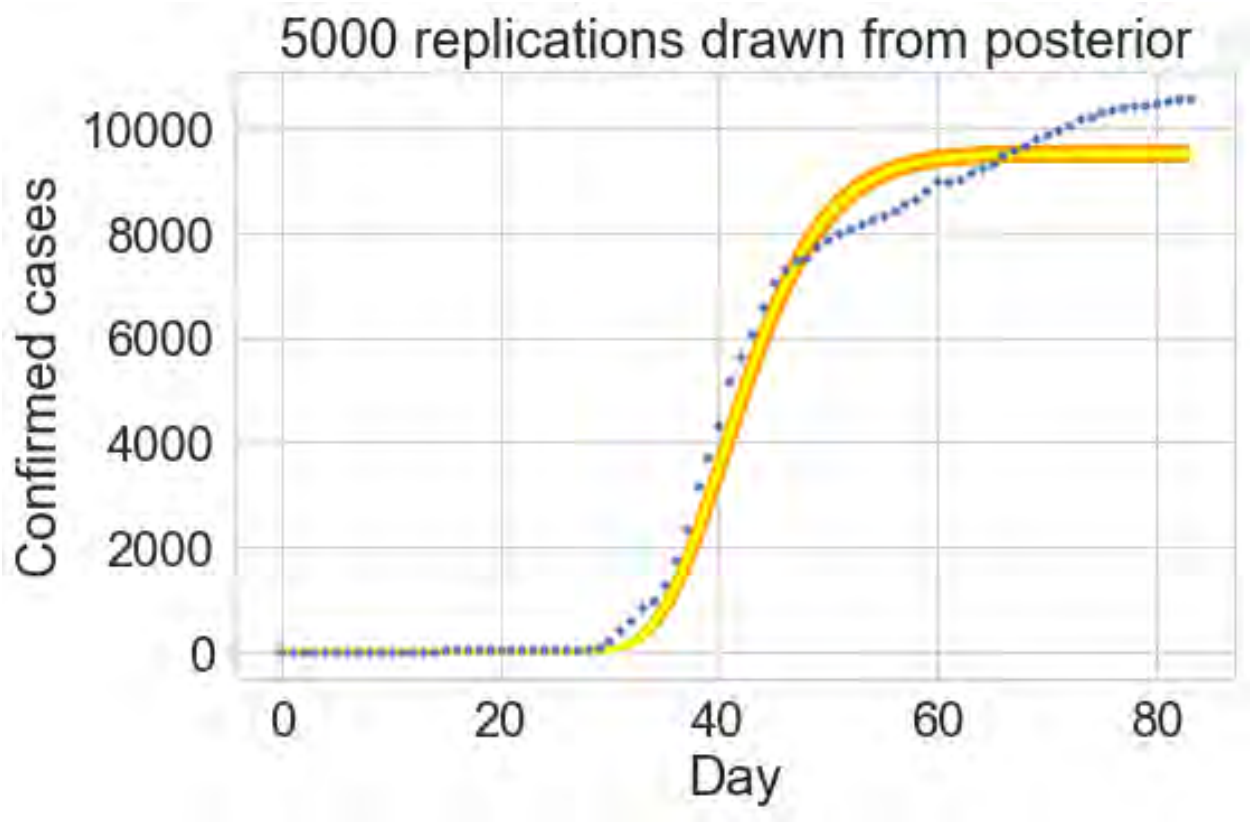


Gaussian kernel with bandwidth = 1.0 is used for a.

Gaussian kernel with bandwidth = 0.1 is used for b.

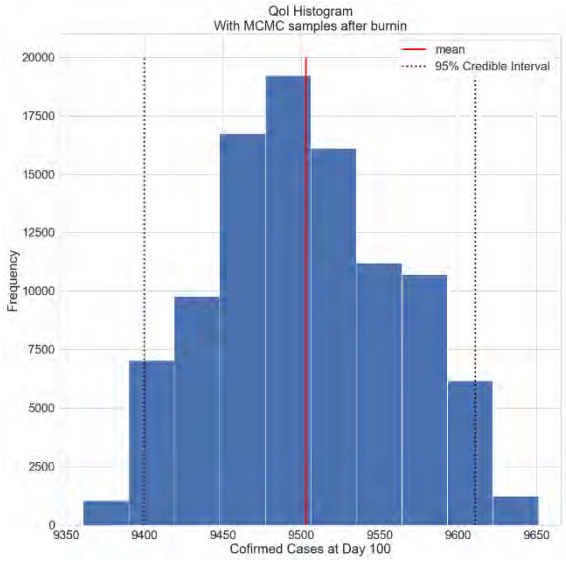
Gaussian kernel with bandwidth = 0.01 is used for p.

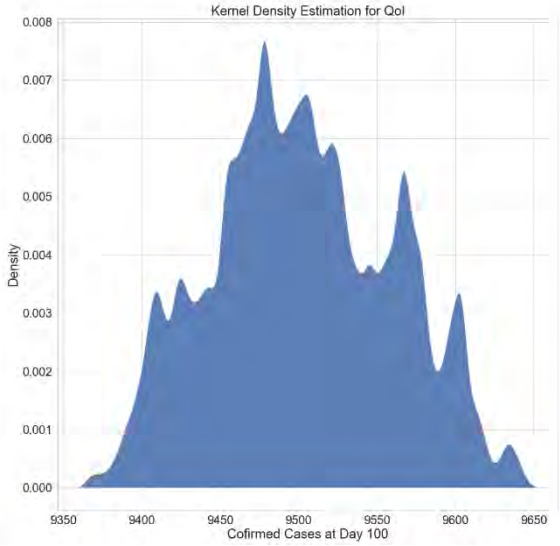
Graphical accuracy check



The validation metric for the model 3, $|C_{m3}(83) - Y_{84}| = 0.1 * Y_{84}$.

Prediction





QoI mean: 9,503

QoI standard deviation: 58

Chapter 4

Predicting COVID19 trends with Markov chain Monte Carlo: A simple exercise of the Metropolis–Hastings algorithm

Jing Hu

May 2020

1 Introduction

Coronavirus disease 2019 (COVID19) is a highly infectious disease first identified in December 2019 in Wuhan, China. It has caused more 3.5 millions of cases and more than 248,000 deaths till now as of 5 May 2020[1]. The rapid spread of the novel disease not only poses a direct threat to people health, it also causes significant economic recessions all around the world. To curb a wider transmission of COVID19, it is essential to predicting the trends of COVID19 in different regions of the world to further make corresponding policies to alleviate its impact. In this report, we make an attempt to deliver a reliable predictive model based on a phenomenological epidemic growth model whose up-to-date parameters are evaluated by constantly assimilating available data through a Markov chain Monte Carlo method (MCMC).

2 Implementation

In this section, we present the implementation of the Metropolis–Hastings algorithm for predicting COVID19 trends in Japan, South Korea and the US. We adopt the following two-parameter generalized-growth model[2]

$$C(t) = \left(\frac{r}{m}t + (C_0)^{1/m}\right)^m \tag{1}$$

where $C(t) : [0, t] \to [0, \infty)]$ is the total confirmed COVID19 cases at time, C_0 is the initial condition and r and m are two parameters we would sampling through the Metropolis–Hastings algorithm. The implementation is illustrated in the following algorithm (1).

In the implementation, we adopt the following distribution function, which is proportional to the posterior probability, as

$$\pi(Y_i|\theta)) \sim e^{-\frac{(Y_i - f(t_i,\theta))^2}{2\sigma^2}} \tag{2}$$

where $f(t_i, \theta)$ is the model outputs at time t_i given parameter $\theta = \{r, m\}$ and we let the model noise $\sigma = 10$. We enforce non-negativity for the prior $\pi(\theta)$ and $\pi(\theta')$ given that no prior knowledge is available in the calibration. In the validation and Bayesian learning process, we evaluate $\pi(\theta)$ and $\pi(\theta')$ making use of the prior evidence and we also assume $\{r, m\}$ follow Gaussian distributions and r and m are independent on each other. We set the burn in rate as 50% in calibration and 25% in daily validation.

In the following we will use COVID19 data at discrete times $\bar{t} = (t_1 = 0, t_2 = 1, ..., t_N = 83)$ where $N = 84$ and and corresponding total confirmed cases $Y(\bar{t}) = (Y_1, Y_2, ..., Y_N)$ from Japan, South Korea and the US to investigate the efficacy of the proposed model. We divide the data in calibration and validation by taking first 70 days data as calibration data and rest data as validation and also Bayesian learning data. We will keep assimilating newly logged data and updating the model parameters on a daily base after calibration.

Algorithm 1: The Metropolis–Hastings algorithm for COVID19 prediction

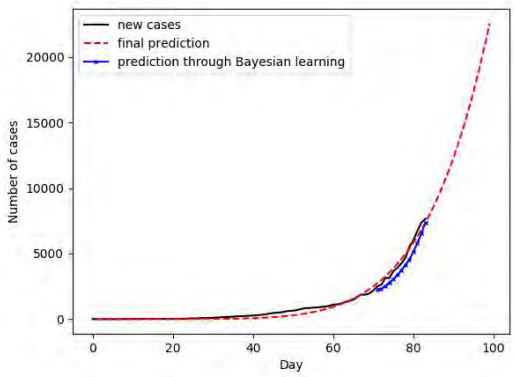
1. calibration;
initialization with guessed $\{r, m\}$;
while *iterations n: training data size* **do**
  generate a candidate through the transition model;
  calculate the acceptance $a = \frac{(\prod_{i=1}^{n} \pi(Y_i|\theta^{'}))\pi(\theta^{'})}{(\prod_{i=1}^{n} \pi(Y_i|\theta))\pi(\theta)}$, where n is calibration data size;
  if $a > 1$ *or a generated uniform random number* $u \in [0, 1]$ *that* $u < a$ **then**
    accept the candidate;
  else
    reject the candidate;
  end
end
2. validation and updating the model by assimilating newly logged data;
while *iterations m: validation data size* **do**
  calculate mean and standard deviation of $\{r, m\}$ of the accepted sample from last step;
  initialize $\{r, m\}$ with the posterior from last step, e.g. mean and standard deviation;
  while *iterations n: training data size* **do**
    generate a candidate through the transition model;
    calculate the acceptance $a = \frac{(\prod_{i=1}^{k} \pi(Y_i|\theta^{'}))\pi(\theta^{'})}{(\prod_{i=1}^{k} \pi(Y_i|\theta))\pi(\theta)}$, where k is prior data size;
    if $a > 1$ *or a generated uniform random number* $u \in [0, 1]$ *that* $u < a$ **then**
      accept the candidate;
    else
      reject the candidate;
    end
  end
end

3 Results and discussion

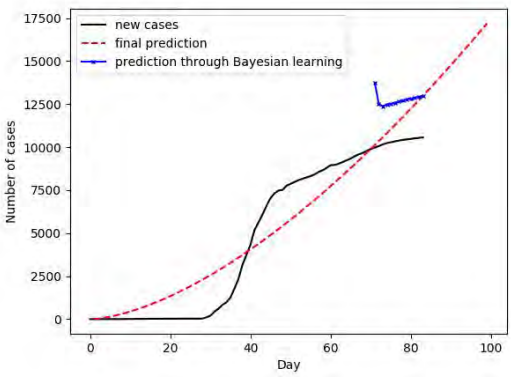
Figure (1) below illustrates the general fit of the predictive models to the number of confirmed cases and the efficacy of the model to infer the near future trends based on prior knowledge for Japan, South Korea and the US. It can be observed that the proposed predictive models are less suitable for South Korea than for Japan and the US, because the epidemic growth model [2] is merely able to capture sub-exponential and exponential epidemic growth, however the outbreak in South Korea mainly displays a "S"-shaped growth. This indicates a more sophisticated model is desired for more extensive epidemic growth of COVID19 in different regions of the world. The deficiency of the model is also reflected in the discouraging performance in inferring near future COVID19 trend in South Korea, as indicated by the blue lines in Figure (1).

Figure (2) below presents the Markov chain marching trajectories in the Metropolis–Hastings algorithm. For all three countries, we let the initial parameters $\{r, m\} = \{1, 1\}$ and we generate 5000 samples for both calibration and each step in validation. As seen in Figure (2), $\{r, m\}$ would gradually localize in the parameter space along the temporal evolution. The model with the localized parameters predicts well the global trend of epidemic growth, however it may not be able to serve as a predictive model if the model cannot capture the development of the event in each period.

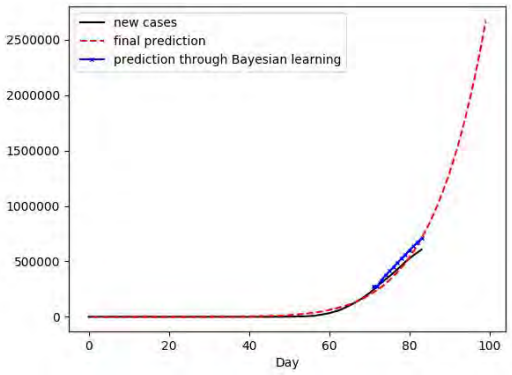
As shown in Figure (3) and (4), the distributions of $\{r, m\}$ of all accepted samples do not meet the Gaussian distribution assumption well. In future work, kernel density estimation (KDE) is desired for further improvement. Besides, a selective stochastic process model would be worthwhile to be considered due to the temporal nature of the epidemic events. Figure (5) presents the joint distribution of r and m. It is clear that r and m are correlated with each other and their correlation could be beneficial to consider to improve the current model.



(a) Japan



(b) South Korea

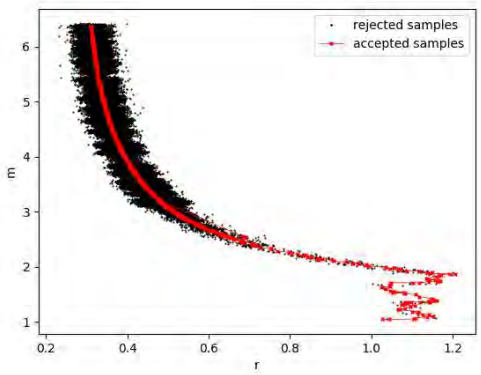


(c) US

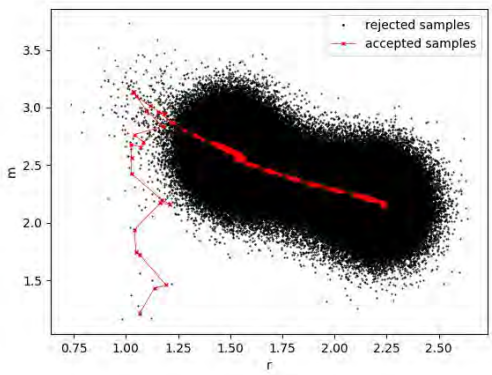
Figure 1: Comparison of predictions with observance: (a) Japan, (b) South Korea and (c) US

4 Conclusion

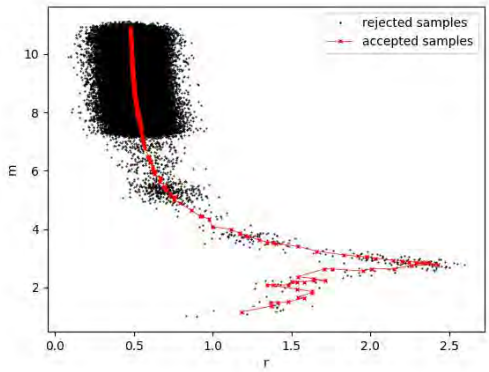
In this report, we attempt to propose a simple and reliable model to predict COVID19 trends in Japan, South Korea and the US, with Markov chain Monte Carlo realized through the Metropolis–Hastings algorithm. The main conclusions of this study can be drawn as follows:
(a) The proposed model performs well when the number of COVID19 cases displays a sub-exponential and exponential epidemic growth form. However for more general epidemic growth, a more developed model is required.
(b) The assumptions made about the parameter distributions are not well-informed in this report. More educated assumptions would be desired to improve the performance of the proposed model.



(a) Japan

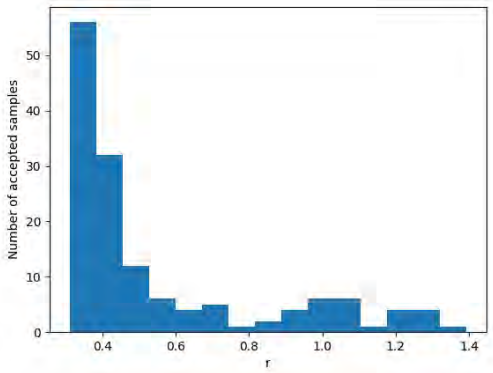


(b) South Korea

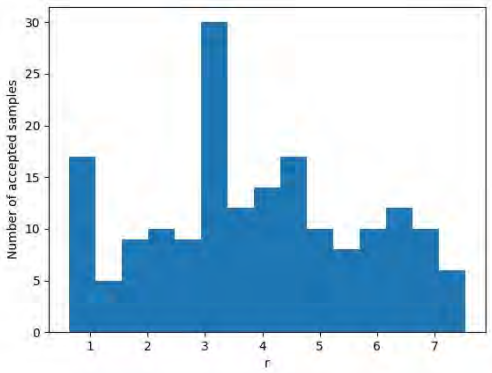


(c) US

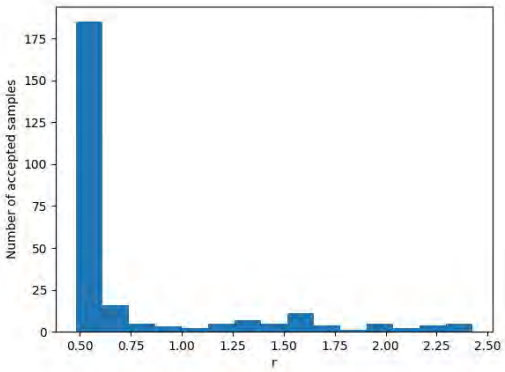
Figure 2: Markov chain marching trajectories: (a) Japan, (b) South Korea and (c) US



(a) Japan

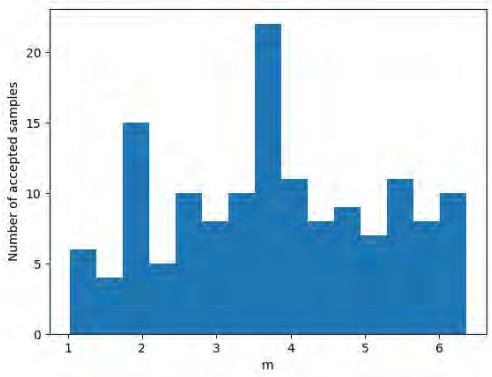


(b) South Korea

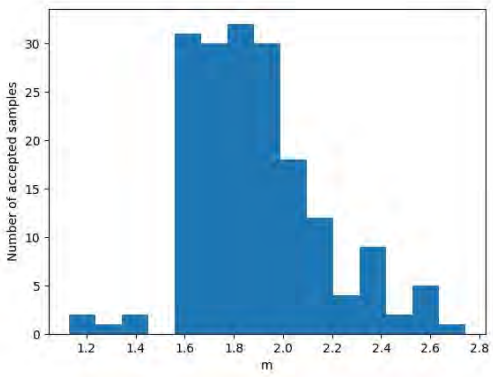


(c) US

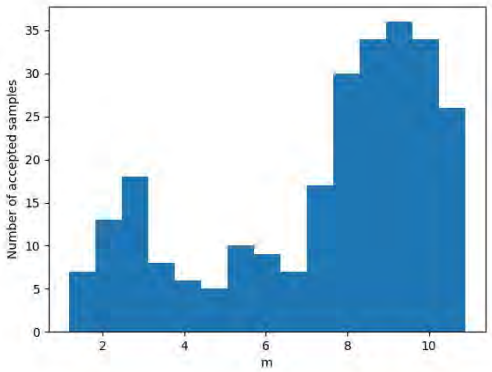
Figure 3: r histogram: (a) Japan, (b) South Korea and (c) US



(a) Japan

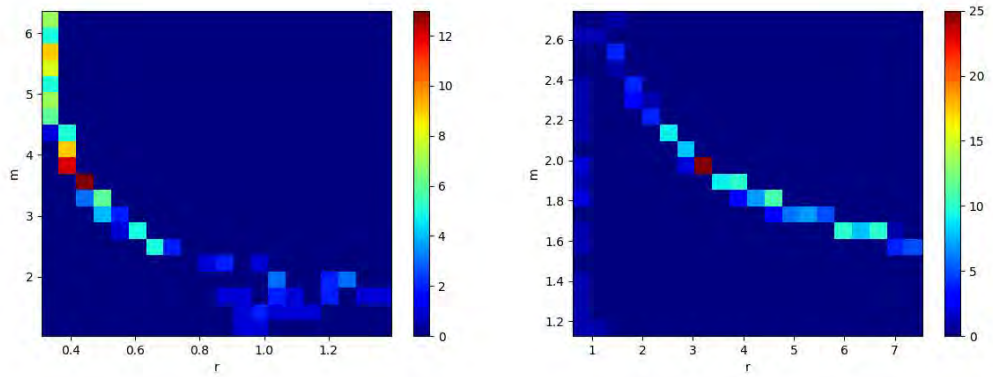


(b) South Korea



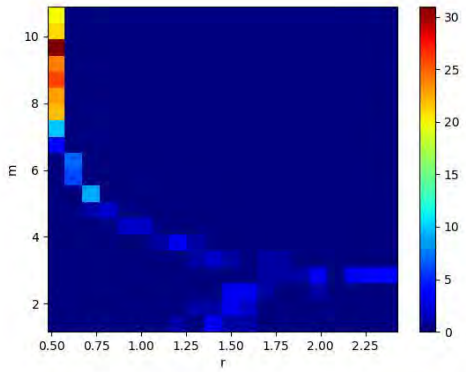
(c) US

Figure 4: r histogram: (a) Japan, (b) South Korea and (c) US



(a) Japan

(b) South Korea



(c) US

Figure 5: $\{r, m\}$ joint distribution: (a) Japan, (b) South Korea and (c) US

A Appendix. code

```
import pandas as pd
import numpy as np
from numpy import genfromtxt
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import scipy.stats

# extract data
df1 = genfromtxt('Japan.csv', delimiter=',')
df2 = genfromtxt('Korea, South.csv', delimiter=',')
df3 = genfromtxt('US.csv', delimiter=',')

# size of calibration data (days)
d=71
# size of total data (days)
dt=84

# prepare training data
df1t=df1[0:d,] # Japan
df2t=df2[0:d,] # South Korea
df3t=df3[0:d,] # US

# define model
pred= lambda x: (r/m*(x)+df3t[0,0]**(1/m))**m

#####################################
# define tranistion model
transition_model = lambda x: np.random.normal(x,[0.1,0.1],(2,))
import math

# prior without any information
def prior(r,m,r_sig,m_sig,w):
    if (abs(r)<0.0000000000001 and abs(m)<0.0000000000001 and abs(r_sig)<0.0000000000001
    ↪  and abs(m_sig)<0.0000000000001): # calibration
          if( w[0]<=0 or w[1] <=0): # enforce nonnegtivity in calibration
              return 0
          else:
              return 1
    else:                              # validation
          return scipy.stats.norm(r, r_sig).pdf(w[0])*scipy.stats.norm(m, m_sig).pdf(w[1])


# calculate log likelihood
def manual_log_lik1(c0,x,data):
    sigma=10.0  # noise
    return
    ↪  np.sum(((x[0]/x[1]*data[:,0]+c0**(1/x[1]))**x[1]-data[:,1])**2.)*(-0.5)/sigma/sigma
#    x[0]: r
#    x[1]: m

# defines whether to accept or reject the new sample
def acceptance(x, x_new):
```

```
    if x_new>x:
        return True
    else:
        accept=np.random.uniform(0,1)

def metropolis_hastings(c0,likelihood_computer,prior, transition_model,
↪  param_init,iterations,data,acceptance_rule,r,m,r_sig,m_sig):
    # likelihood_computer(x,data): returns the likelihood that these parameters generated
    ↪  the data
    # transition_model(x): a function that draws a sample from a symmetric distribution
    ↪  and returns it
    # param_init: a starting sample
    # iterations: number of samples to generat
    # data: the data that we wish to model
    # acceptance_rule(x,x_new): determine whether to accept or reject the new sample
    x = param_init
    accepted = []
    rejected = []
    for i in range(iterations):
        x_new =  transition_model(x)
        x_lik = likelihood_computer(c0,x,data)
        x_new_lik = likelihood_computer(c0,x_new,data)
        if (acceptance_rule(x_lik +
        ↪  np.log(prior(r,m,r_sig,m_sig,x)),x_new_lik+np.log(prior(r,m,r_sig,m_sig,x_new))
        ↪  )):
            x = x_new
            accepted.append(x_new)
        else:
            rejected.append(x_new)

    return np.array(accepted), np.array(rejected)


#==========================================================================
# Calibration
r=0.0
m=0.0
r_sig=0.0
m_sig=0.0

# Viboud et al 2015. formula
accepted, rejected =
↪  metropolis_hastings(df3t[0,0],manual_log_lik1,prior,transition_model,[1, 1],
↪  5000,df3t,acceptance,r,m,r_sig,m_sig)
acceptedall=accepted
rejectedall=rejected

# truncate burn in
temp=np.array(accepted.shape)
accepted1=accepted[int(round(temp[0]/2)):,]

# calculate {r,m} averge for initialization
r=np.average(accepted1[:,0])
m=np.average(accepted1[:,1])
r_sig=np.std(accepted1[:,0])
```

```
m_sig=np.std(accepted1[:,1])


# calculate the difference between validation data and observance
pred_array=[0]*(dt-d)
pred_array[0]=pred(d)

#================================================================================
# Validation and Bayesian learning
# conduct validation and updates {r,m} through MCMC
print('start validation process')
print('========================================')
for i in range(dt-d-1):
      # Prepare training data
      df1t=df1[0:(d+i+1),] # Japan
      df2t=df2[0:(d+i+1),] # South Korea
      df3t=df3[0:(d+i+1),] # US

      print(i,'==',(d+1+i),'==',acceptedall.shape,'==',accepted.shape)
      print(r,m,r_sig,m_sig)

      # Viboud et al 2015. formula
      accepted, rejected =
      ↪  metropolis_hastings(df3t[0,0],manual_log_lik1,prior,transition_model,[r, m],
      ↪  5000,df3t,acceptance,r,m,r_sig,m_sig)
      temp=np.array(accepted.shape)
      if (temp[0]==0):
            acceptedtemp=acceptedall
      else:
            acceptedtemp=np.concatenate((acceptedall, accepted))

      acceptedall=acceptedtemp
      rejectedtemp=np.concatenate((rejectedall, rejected))
      rejectedall=rejectedtemp

      # truncate burn in
      temp=np.array(accepted.shape)
      accepted1=accepted[int(round(temp[0]/4)):,]

      # calculate {r,m} averge for initialization
      if (temp[0]!=0):
            r=np.average(accepted1[:,0])
            m=np.average(accepted1[:,1])
            r_sig=np.std(accepted1[:,0])
            m_sig=np.std(accepted1[:,1])

      # calculate the difference between validation data and observance
      pred_array[(i+1)]=pred(d+i+1)

#================================================================================
# post-processing
# plot Markov chain tranjectory
fig = plt.figure(1)
line1,=plt.plot(rejectedall[:,0],rejectedall[:,1],'o',
↪  color='black',markersize=0.5,label='rejected samples')
```

```
line2,=plt.plot(acceptedall[:,0],acceptedall[:,1],'-x',
↪  color='red',markersize=2.5,label='accepted samples',linewidth=0.5)
plt.xlabel('r')
plt.ylabel('m')
lgnd=plt.legend(handles=[line1, line2])
lgnd.legendHandles[0]._legmarker.set_markersize(1)
lgnd.legendHandles[1]._legmarker.set_markersize(3)
plt.savefig('USPresultplot', dpi=fig.dpi)

# plot prediction and evidence
fig = plt.figure(2)
line3,=plt.plot(df3[:,0],df3[:,1],'k',label='new cases')
line4,=plt.plot(np.arange(1,100,1),pred(np.arange(1,100,1)),'r--',label='final
↪  prediction')
line5,=plt.plot(np.array(np.arange(d, dt,
↪  1)),np.array(pred_array),'-x',color='b',markersize=2.5,label='prediction through
↪  Bayesian learning')

lgnd=plt.legend(handles=[line3, line4, line5])
plt.xlabel('Day')
plt.ylabel('Number of cases')
plt.savefig('USplot', dpi=fig.dpi)

# plot {r,m} histogram
fig = plt.figure(3)
plt.hist(acceptedall[:,0], bins=15)
plt.xlabel('r')
plt.ylabel('Number of accepted samples')
plt.savefig('US_r_hist', dpi=fig.dpi)

fig = plt.figure(4)
plt.hist(acceptedall[:,1], bins=15)
plt.xlabel('m')
plt.ylabel('Number of accepted samples')
plt.savefig('US_m_hist', dpi=fig.dpi)

fig= plt.figure(5)
plt.hist2d(acceptedall[:,0],acceptedall[:,1],bins=(20, 20),cmap=plt.cm.jet)
plt.colorbar()
plt.xlabel('r')
plt.ylabel('m')
plt.savefig('US_rm_hist', dpi=fig.dpi)
```

References

[1] Johns Hopkins University. Covid-19 dashboard by the center for systems science and engineering (csse) at johns hopkins university (jhu). https://gisanddata.maps.arcgis.com/apps/opsdashboard/index.html/bda7594740fd40299423467b48e9ecf6, Retrieved 4 May 2020.

[2] Cécile Viboud, Lone Simonsen, and Gerardo Chowell. A generalized-growth model to characterize the early ascending phase of infectious disease outbreaks. *Epidemics*, 15:27–37, 2016.

Chapter 5

Fitting COVID19 trends using Bayesian method

by Mathew Hu

In this assignment, our goal is to fit the COVID 19 trends (confirmed cases of COVID 19 as a function of date) using the generalized growth model given by

$$\frac{dC(t)}{dt} = rC(t)^p, \tag{1}$$

where $t \in [0, T]$ is the time (in units of days), $r \geq 0$ is the growth rate, $p \in [0, 1]$ is the 'decceleration of growth' parameter, see **Viboud et al 2015**. Special cases: $p = 0$ gives linear growth model and $p = 1$ gives exponential growth model.

$C : [0, T] \to [0, \infty)$ is the total confirmed COVID 19 cases at time t. When $0 < p < 1$, Eq (1) can be solved to get

$$C(t) = \left(\frac{r}{m}t + (C_0)^{1/m}\right)^m, \tag{2}$$

where $m = 1/(1 - p)$ and $C_0 = C(0)$ is the initial condition. For special cases $p = 0$ and $p = 1$, C can be found easily.

- When $p = 0$, $C = C_0 + rt$
- When $p = 1$, $C = C_0 \exp[rt]$

Alternative model

Another model is given as

$$C(t) = a \exp\left[b\left(1 - \frac{1}{1 - (1 - t/T)^p}\right)\right] + C_0, \tag{3}$$

where a, b are model parameters and $p > 1$ is the fixed exponent.

1. Problem

We cosider COVID 19 data at discrete times $\bar{t} = (t_1 = 0, t_2 = 1, \ldots, t_N = 83)$, where $N = 84$, and corresponding total confirmed cases $Y(\bar{t}) = (Y_1, Y_2, \ldots, Y_N)$.

Model prediction is $C(\bar{t}) = (C(t_1), C(t_2), \ldots, C(t_N))$ where $C(t_i)$ is given by Eq (2). The model parameters are $\theta = (r, p)$. Take uniform prior for θ and consider a Guassian noise with zero mean and standard deviation σ. Divide the data in calibration and validation by taking first $N_c = 50$ as the calibration data and rest $N_v = N - N_c$ as validation data. You can also try different priors for parameters and divide data differently in calibration and validation set.

Problem: Predict the total confirmed cases at $T = 100$ day for three countries US, Japan, and South Korea.

1.1 Data

Data for current epidemic COVID 19 can be found in several places such as:

- datasets/covid-19 (https://github.com/datasets/covid-19)
- CSSEGISandData/COVID-19 (https://github.com/CSSEGISandData/COVID-19)
- nytimes/covid-19-data (https://github.com/nytimes/covid-19-data)

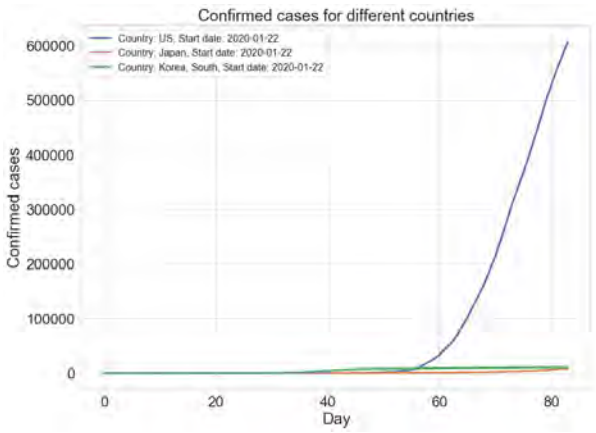
In StudyCovid19 (https://github.com/prashjha/StudyCovid19) you will find python scripts to process data and this notebook in directory `process/bayesian` .

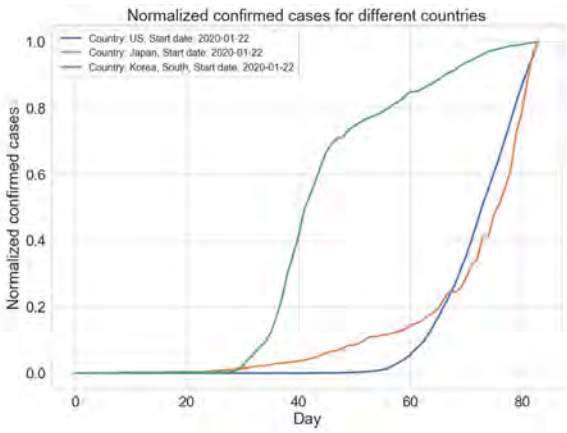
Below we plot the confirmed cases for various countries

In [1]:
```
import pandas as pd
import sys
sys.path.insert(0,'../')
```

In [15]:
```
from data import plot_countries_all_plus_normalize_1 as plot
from data import save_country_data_1 as save
df = pd.read_csv('../../data/datasets/time-series-19-covid-combined.csv'
)
# plot
plot(df, ['US', 'Japan', 'Korea, South'])
# save data
save(df, ['US', 'Japan', 'Korea, South'])
```

```
Num days: 84
```





2. Inplement Bayessian method

Take South Korea as an example. Following we will apply the calibration and validation, and give QoI(#Confirmed case at T=100)'s expectation and distribution.

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
from numpy import linalg as LA
from scipy.stats import multivariate_normal
from models import model1, model3
T = 100
```

2.1 Preparation

Given data of a specific country, we fist need to split it into a training set and a testing set. My approach is to set 80% of the data as training, and the rest 20% as testing. Also, we need to specify a forward model which takes some parameters as input and returns the numbers of confirmed cases as its output. The models are aready given above.

To apply the bayessian rule we need a prior and a likelihood function. Here we give prior as an uniform distribution, which implies that we know nothing about the parameters initially. And here we give likelihood as an Gaussian distribution, which is because we suppose a Gaussian noise for the data.

In [4]:
```
# prepare data, model, and the corresponding prior and likelihood
def prepare_data(country):
    # open file
    df = pd.read_csv(country+'.csv',header=None)
    data = np.array(df)
    # split into calibration and validation sets
    train_size = int(data.shape[0] * 0.8)
    data_c = data[:train_size,:]
    data_v = data[train_size:,:]
    return data_c, data_v

def prepare_method(model,country,data,noise_arg):
    # prepare model 1
    if model == model1:
        def log_prior_c(theta):
            if theta[0]<0 or theta[1]<0 or theta[1]>1:   # r>0  0<=p<=1
                return -inf
            return 0
        # arguments used in the model
        fix_params = [data[0,1]]
        trans_param = [0.005,0.0005]
        param_init = [0.5,0.5]

    # prepare model 3
    if model == model3:
        # uniform prior
        def log_prior_c(theta):
            if theta[0]<0 or theta[1]<0:   # a>0,b>0
                return -np.inf
            return 0
        # arguments used in the model
        fix_params = [data[0,1],T,4]
        trans_param = [noise_arg,.001*noise_arg]
        if country=='Korea, South':
            param_init = [12000,8]
        if country=='US':
            param_init = [480000,100]
        if country=='Japan':
            param_init = [1000,8]
    return log_prior_c, fix_params, trans_param, param_init

def log_likelihood(theta, data):
    tdata = data[:,0]
    ydata = data[:,1]
    return np.sum( - np.log(noise_arg * np.sqrt(2*np.pi))
                   - ((ydata-model(theta,tdata,fix_params))**2)/(2*noise
_arg**2) )
```

In addition, we will use MCMC to draw samples from the posterior we derive. Here we inplement the Metropolis–Hastings algorithm.

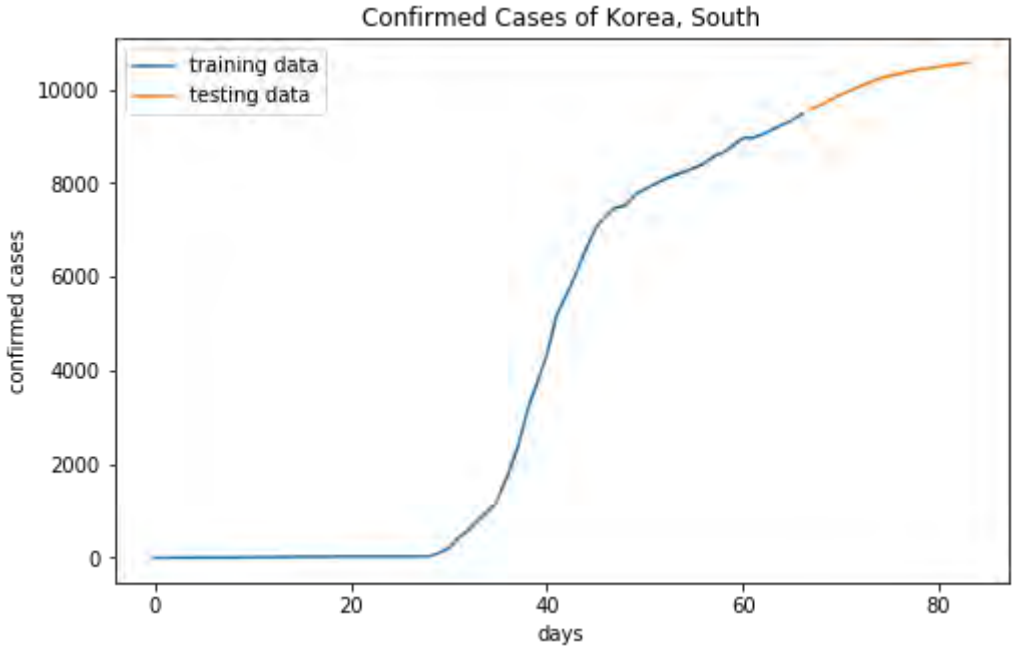
In [5]:
```
# prepare MCMC
def acceptance_rule(x, x_new):     # acceptance rule under log density
    if x_new>x:
        return True
    else:
        accept = np.random.uniform(0,1)
        # Since we did a log likelihood, we need to exponentiate in orde
r to compare to the random number
        # less likely x_new are less likely to be accepted
        return (accept < (np.exp(x_new-x)))

def metropolis_hastings(log_likelihood,log_prior, transition_model, para
m_init,iterations,data,acceptance_rule):
    # likelihood_computer(x,data): returns the likelihood that these par
ameters generated the data
    # transition_model(x): a function that draws a sample from a symmetr
ic distribution and returns it
    # param_init: a starting sample
    # iterations: number of accepted to generated
    # data: the data that we wish to model
    # acceptance_rule(x,x_new): decides whether to accept or reject the
 new sample
    theta = param_init
    accepted = []
    rejected = []
    for i in range(iterations):
        theta_new = transition_model(theta)
        lik = log_likelihood(theta,data)
        lik_new = log_likelihood(theta_new,data)
        if (acceptance_rule(lik + log_prior(theta),lik_new + log_prior(t
heta_new))):
            theta = theta_new
            accepted.append(theta_new)
        else:
            rejected.append(theta_new)
    return np.array(accepted), np.array(rejected)

def show_mcmc(accepted, rejected):
    fig = plt.figure(figsize=(10,6))
    ax = fig.add_subplot(1,1,1)
    ax.plot(accepted[:,0], accepted[:,1], label="Path")
    ax.plot(accepted[:,0], accepted[:,1], 'b.', label='Accepted',alpha=
0.3)
    ax.plot(rejected[:,0], rejected[:,1], 'rx', label='Rejected',alpha=
0.3)
    ax.set_xlabel("first parameter")
    ax.set_ylabel("second parameter")
    ax.legend()
    ax.set_title("MCMC sampling with Metropolis-Hastings. All samples ar
e shown.")

def burn_accepted(accepted, burn_in=0.8, show=True):
    total_iters = accepted.shape[0]
    burnin_iters = int(burn_in*accepted.shape[0])
    accepted_burnin = accepted[:burnin_iters,:]
    accepted_burned = accepted[burnin_iters:,:]
```

```
    if show:
        for i in range(2):
            fig = plt.figure(figsize=(15,7))
            ax = fig.add_subplot(1,2,1)
#             ax.plot(range(burnin_iters),accepted_burnin[:,0], color='g
ray', label='burn_in')
            ax.plot(range(burnin_iters,total_iters),accepted_burned[:,i
], color='blue', label='burned')
            ax.set_title("Trace for #{} parameter (burned)".format(i+1))
            ax.set_xlabel("Iteration")
            ax.set_ylabel("parameter")
            ax.legend()
            ax = fig.add_subplot(1,2,2)
            ax.hist(accepted_burned[:,i], bins=20, density=True)
            ax.set_ylabel("Frequency (normed)")
            ax.set_xlabel("parameter")
            ax.set_title("Histogram of #{} parameter (burned)".format(i+
1))
            fig.tight_layout()
    return accepted_burned
```

2.2 Let's start with a country and a method!

In [6]:
```
country = 'Korea, South'
model = model3
data_c, data_v = prepare_data(country)
noise_arg = .001 * data_c[-1,1]
print('noise standard deviation = ', noise_arg)
log_prior_c, fix_params, trans_param_c, param_init_c = prepare_method(mo
del,country,data_c,noise_arg)

plt.figure(figsize=(8,5))
plt.plot(data_c[:,0],data_c[:,1], label="training data")
plt.plot(data_v[:,0],data_v[:,1], label="testing data")
plt.xlabel("days")
plt.ylabel("confirmed cases")
plt.legend()
plt.title("Confirmed Cases of "+ country)
plt.show()
```

```
noise standard deviation =  9.478
```



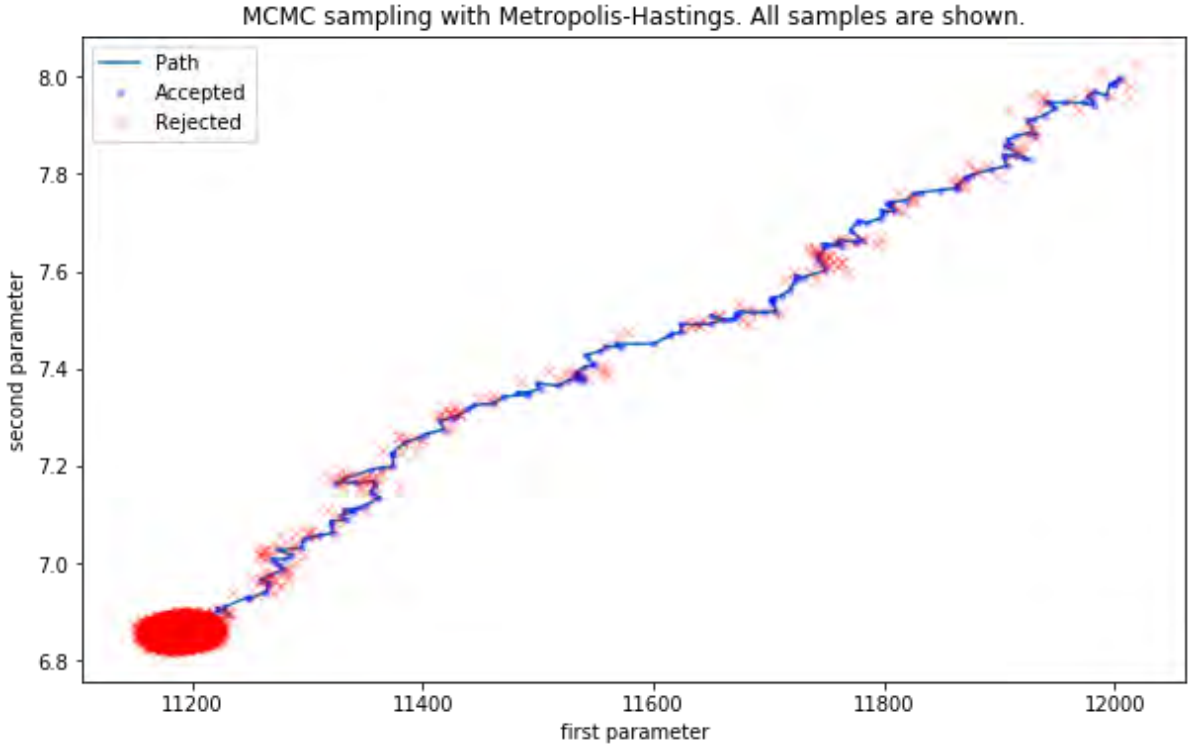
2.3 Calibration

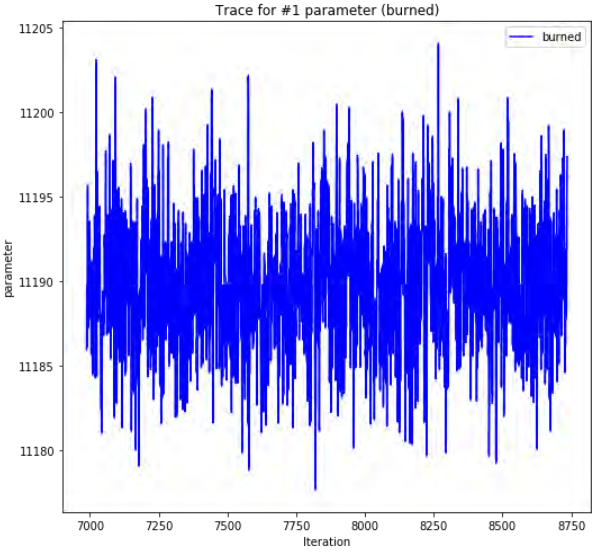
In [7]:
```
transition_model_c = lambda theta: np.random.normal(theta,trans_param_c,
(2,))
accepted_c, rejected_c = metropolis_hastings(log_likelihood,log_prior_c,
transition_model_c,
                                             param_init_c,50000,data_c,a
cceptance_rule)
print('# accepted = ' + str(accepted_c.shape[0]))
```

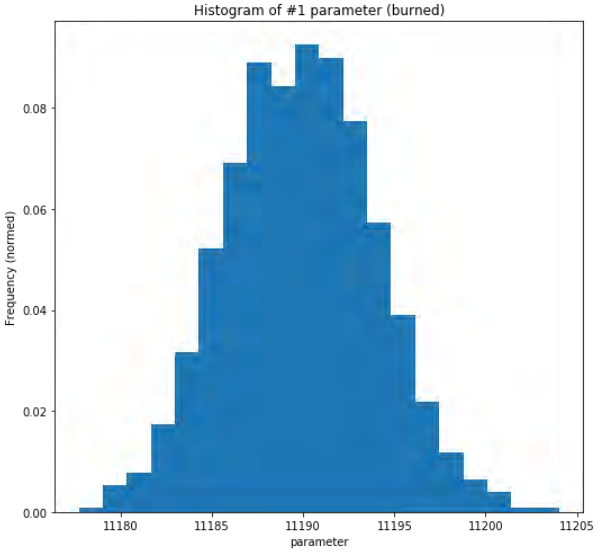
```
# accepted = 8736
```

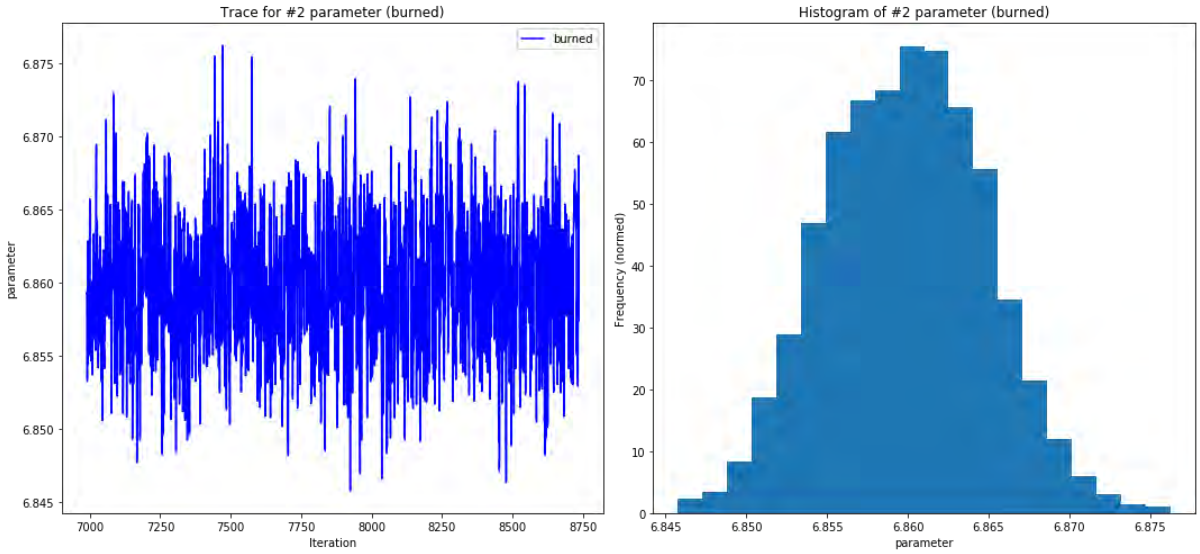
In [8]:
```
show_mcmc(accepted_c, rejected_c)
accepted_burned_c = burn_accepted(accepted_c, burn_in=0.8, show=True)

mean_c = np.mean(accepted_burned_c, axis=0)
cov_c = np.cov(accepted_burned_c.T)
print(mean_c)
print(cov_c)
```

```
[1.11898787e+04 6.85988185e+00]
[[1.64794278e+01 1.35641516e-02]
 [1.35641516e-02 2.43714818e-05]]
```









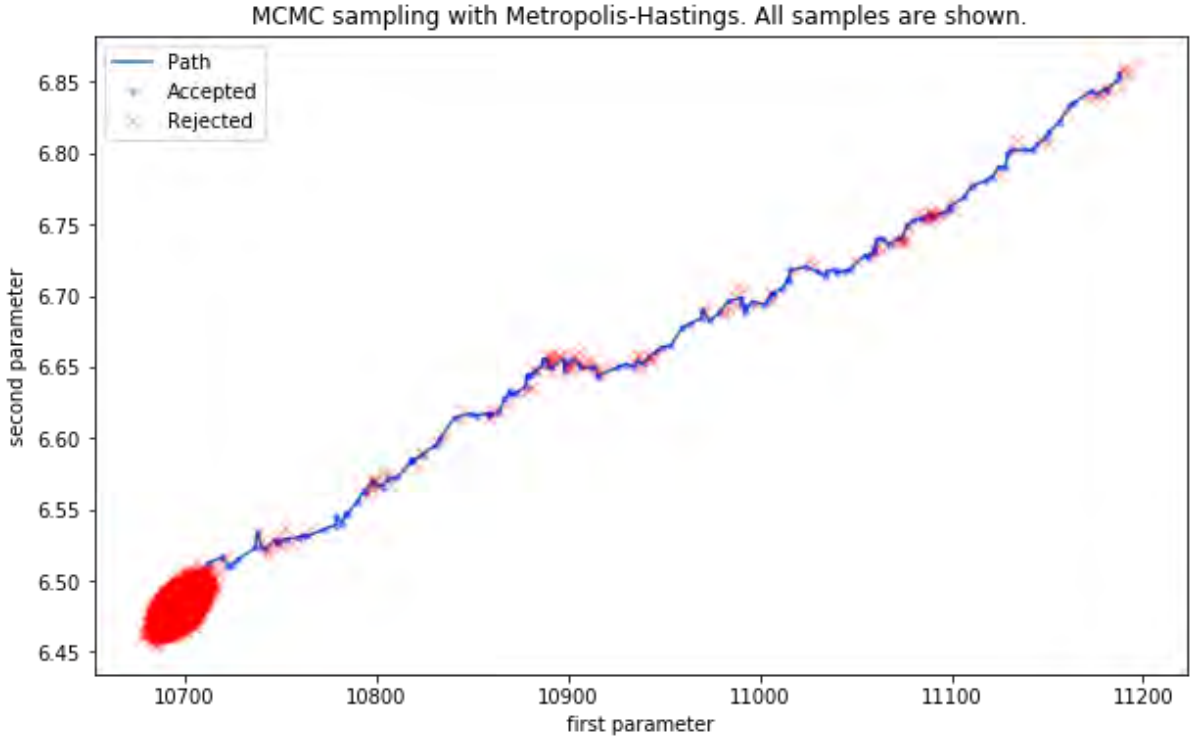
2.4 Validation

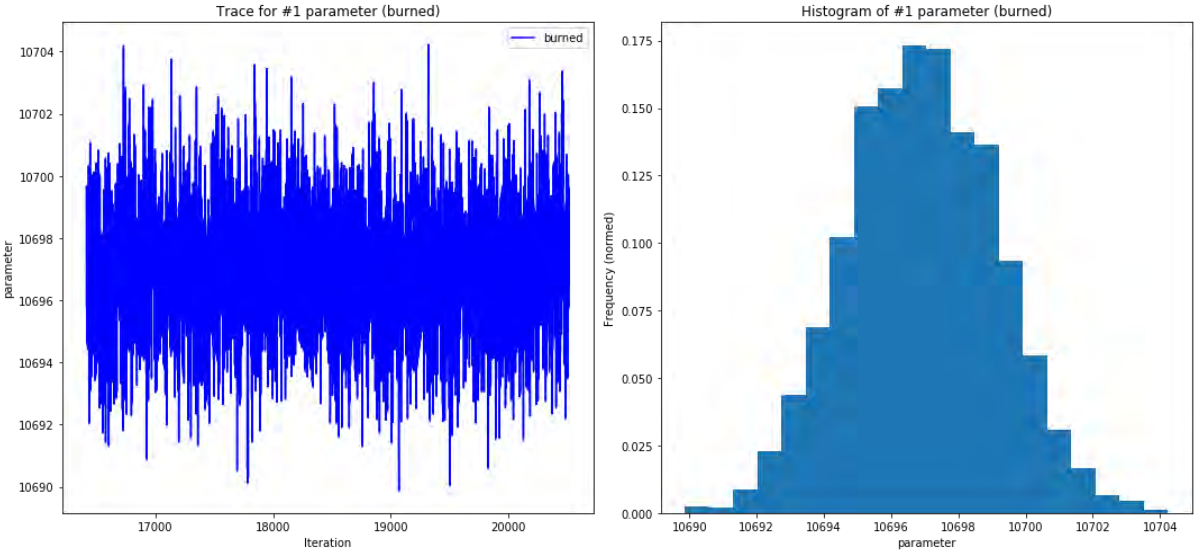
In [9]:
```
param_init_v = mean_c
def log_prior_v(theta):
    return multivariate_normal.logpdf(theta, mean=mean_c, cov=cov_c) # a
pproximate posterior by normal distributon
transition_model_v = lambda theta: multivariate_normal.rvs(mean=theta, c
ov=cov_c)
# transition_model_v = lambda theta: np.random.normal(theta,trans_param_
c,(2,))
accepted_v, rejected_v = metropolis_hastings(log_likelihood,log_prior_v,
transition_model_v,
                                             param_init_v,50000,data_v,a
cceptance_rule)
print('# accepted = ' + str(accepted_v.shape[0]))
```

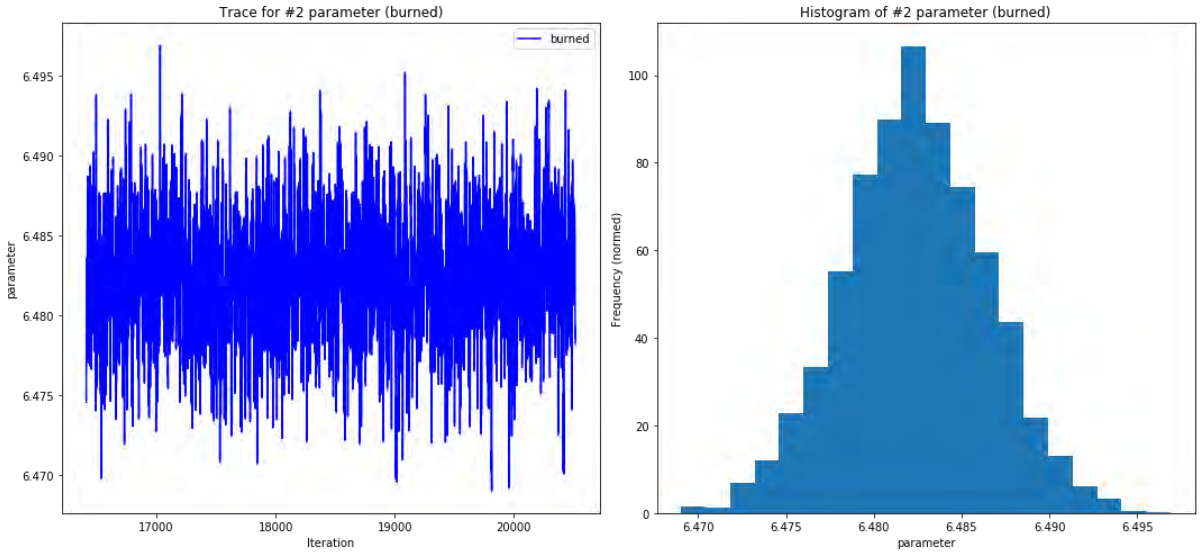
```
# accepted = 20516
```

In [10]:
```
show_mcmc(accepted_v, rejected_v)
accepted_burned_v = burn_accepted(accepted_v, burn_in=0.8, show=True)

mean_v = np.mean(accepted_burned_v, axis=0)
cov_v = np.cov(accepted_burned_v.T)
print(mean_v)
print(cov_v)
```

```
[1.06969164e+04 6.48237556e+00]
[[4.81635227e+00 4.35802594e-03]
 [4.35802594e-03 1.65535668e-05]]
```







To validate the model, we compute the l^2 distance between the prediction on validation scenarios and the observed data

$$d(\mathbf{d}(\theta^{**}, S_v), y_v) = \mathbf{E}_{\theta^{**}} \|\mathbf{d}(\theta^{**}, S_v) - y_v\|_2 = \mathbf{E}_{\theta^{**}} (\sum_{t_i \in S_v} |C(t_i, \theta^{**}) - y(t_i)|^2)^{1/2}$$

where θ^{**} is the parameter updated in validation, which is ramdon and subjected to a posterior we derived.

Furthermore, we have this distance devided by $\|y_v\|_2 = (\sum_{t_i \in S_v} |y(t_i)|^2)^{1/2}$ as a normalization.

In summary, if the normalized error is smaller than a tolerance, then we say that the model is valid.

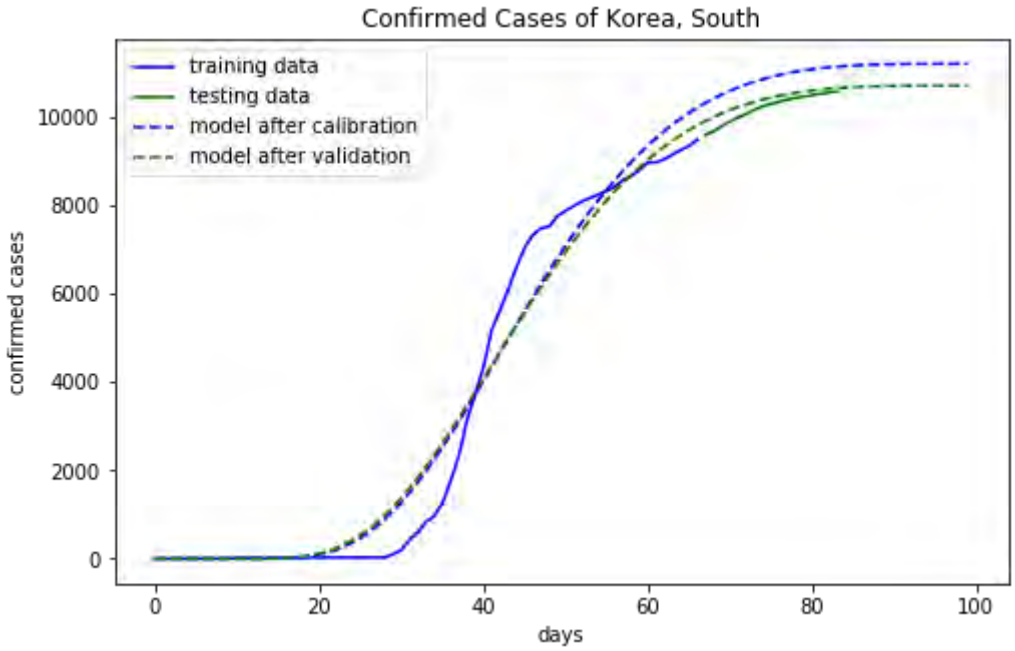
$$d_{normalized}(\mathbf{d}(\theta^{**}, S_v), y_v) = \frac{d(\mathbf{d}(\theta^{**}, S_v), y_v)}{\|y_v\|_2} < \gamma_{tol}$$

In [11]:
```
tol_v = .05
dists = [LA.norm(data_v[:,1]-model(theta,data_v[:,0],fix_params)) for th
eta in accepted_burned_v]
dist = sum(dists)/len(dists)
vad = dist / LA.norm(data_v[:,1])
print('normalized error = ', vad)
print('tolerance = ', tol_v)
if vad<tol_v:
    print('The model is valid.')
else:
    print('The model is invalid.')
```

```
normalized error =  0.018813350806337722
tolerance =  0.05
The model is valid.
```

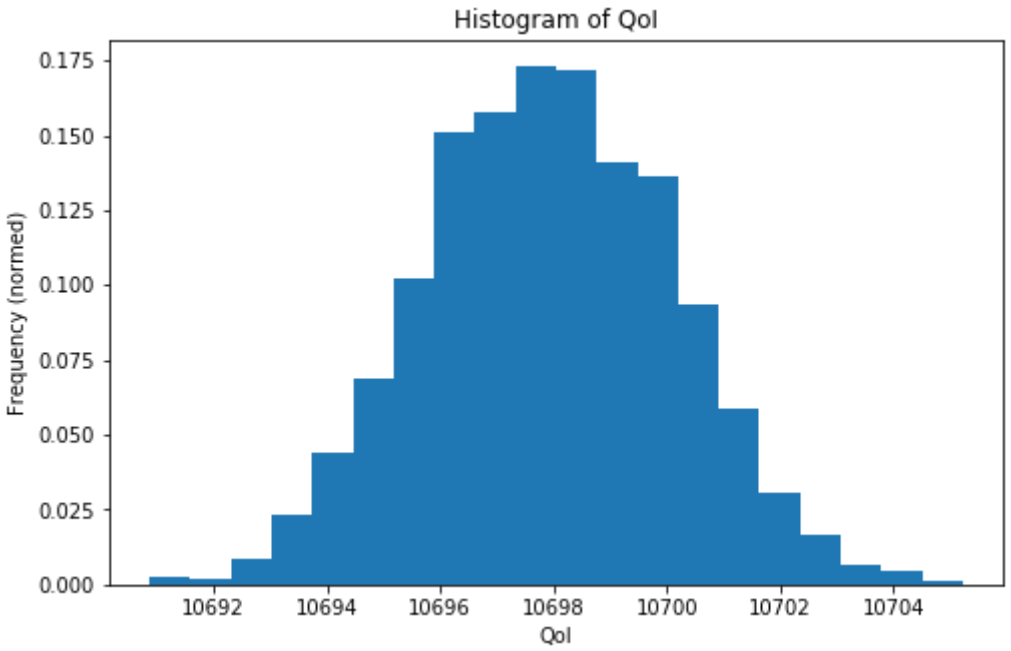
2.5 Show fitting curves and QoI

In [12]:
```
fig = plt.figure(figsize=(8,5))

# plot the curve
plt.plot(data_c[:,0],data_c[:,1], 'b-', label="training data")
plt.plot(data_v[:,0],data_v[:,1], 'g-', label="testing data")
plt.plot(np.arange(T), model(mean_c, np.arange(T), fix_params), 'b--', l
abel="model after calibration")
plt.plot(np.arange(T), model(mean_v, np.arange(T), fix_params), 'g--', l
abel="model after validation")

plt.xlabel("days")
plt.ylabel("confirmed cases")
plt.legend()
plt.title("Confirmed Cases of "+ country)
plt.show()
```



In [13]:
```
QoIs = [model(theta,[100],fix_params)[0] for theta in accepted_burned_v]

fig = plt.figure(figsize=(8,5))
ax = fig.add_subplot(1,1,1)
ax.hist(QoIs, bins=20, density=True)
ax.set_ylabel("Frequency (normed)")
ax.set_xlabel("QoI")
ax.set_title("Histogram of QoI")

QoI = sum(QoIs)/len(QoIs)
print('We expect the number of confirmed cases at T=100 is about', QoI)
```

```
We expect the number of confirmed cases at T=100 is about 10697.916352761991
```



3. Compare models

3.1 By errors in validation

We have errors computed in validation, so we can compare the errors derived with different models.

In [14]:
```
country = 'Korea, South'
data_c, data_v = prepare_data(country)
noise_arg = .001 * data_c[-1,1]
errs = []
for model in [model1, model3]:
    log_prior_c, fix_params, trans_param_c, param_init_c = prepare_metho
d(model,country,data_c,noise_arg)


    transition_model_c = lambda theta: np.random.normal(theta,trans_para
m_c,(2,))
    accepted_c, rejected_c = metropolis_hastings(log_likelihood,log_prio
r_c,transition_model_c,
                                                 param_init_c,50000,data
_c,acceptance_rule)
    accepted_burned_c = burn_accepted(accepted_c, burn_in=0.8, show=Fals
e)
    mean_c = np.mean(accepted_burned_c, axis=0)
    cov_c = np.cov(accepted_burned_c.T)
    param_init_v = mean_c
    def log_prior_v(theta):
        return multivariate_normal.logpdf(theta, mean=mean_c, cov=cov_c)
# approximate posterior by normal distributon
    transition_model_v = lambda theta: multivariate_normal.rvs(mean=thet
a, cov=cov_c)
    accepted_v, rejected_v = metropolis_hastings(log_likelihood,log_prio
r_v,transition_model_v,
                                                 param_init_v,50000,data
_v,acceptance_rule)
    accepted_burned_v = burn_accepted(accepted_v, burn_in=0.8, show=Fals
e)
    dists = [LA.norm(data_v[:,1]-model(theta,data_v[:,0],fix_params)) fo
r theta in accepted_burned_v]
    dist = sum(dists)/len(dists)
    err = dist / LA.norm(data_v[:,1])
    errs.append(err)
print(errs)
if errs[0]<errs[1]:
    print('Model 1 is better.')
else:
    print('Model 3 is better.')
```

```
[0.0912322339603141, 0.019728819625629473]
Model 3 is better.
```

References

- **Viboud et al 2015** Viboud, C., Simonsen, L. and Chowell, G., 2016. A generalized-growth model to characterize the early ascending phase of infectious disease outbreaks. Epidemics, 15, pp.27-37.

Chapter 6

Jon Kelley

CSE 397

Dr. Oden

May 1, 2020

Bayesian Parameter Estimation for COVID-19 Models

I. Model Selection

In order to estimate the number of cases in Japan on the 100th day (our QoI), we will use the model:

$$C(t) = \left(\frac{r}{m}t + C_o^{\frac{1}{m}}\right)^m, m = \frac{1}{1-p}$$

where *t* is the number of days since the outbreak, C_o is the number of positive cases on day zero, and *r* and *p* are the parameters to be fit. In order to fit the model parameters to the data, we will use the Metropolis-Hastings (MH) algorithm.

II. Calibration

For the calibration step, we assume that *p* is drawn from the uniform distribution between $(0,1)$ and *r* is drawn from the uniform distribution between $(0,10)$. Let $\boldsymbol{\theta} = \{r, p\}$ and denote our uniform prior for the calibration step as $\pi(\boldsymbol{\theta})$. Then we estimate the posterior distribution for the calibration data as

$$\pi(\boldsymbol{\theta}|\boldsymbol{y_c}, S_c) \propto \pi(\boldsymbol{y_c}|\boldsymbol{\theta}, S_c)\, \pi(\boldsymbol{\theta}),$$

where $\boldsymbol{y_c}$ is the calibration data, S_c denotes the calibration scenario, $\pi(\boldsymbol{\theta}|\boldsymbol{y_c}, S_c)$ is the calibration posterior and $\pi(\boldsymbol{y_c}|\boldsymbol{\theta}, S_c)$ is the calibration likelihood. If we assume that the noise for the data is normally distributed, we can write the likelihood for the calibration as

$$\pi(\boldsymbol{y_c}|\boldsymbol{\theta}, S_c) \propto \exp\left(-\frac{\|\boldsymbol{y_c} - C(\boldsymbol{\theta})\|_2^{\,2}}{2\sigma^2}\right)$$

where σ is the standard deviation of the noise associated with $\boldsymbol{y_c}$. Upon running MH for one thousand iterations and the cases for $t \in [0,53]$ as calibration data, we find the following distributions for the accepted *r* and *p* values (Fig. 1).

Using the computed means for *r* and *p* (μ_r and μ_p respectively), Figure 2 shows the prediction compared to the calibration data.

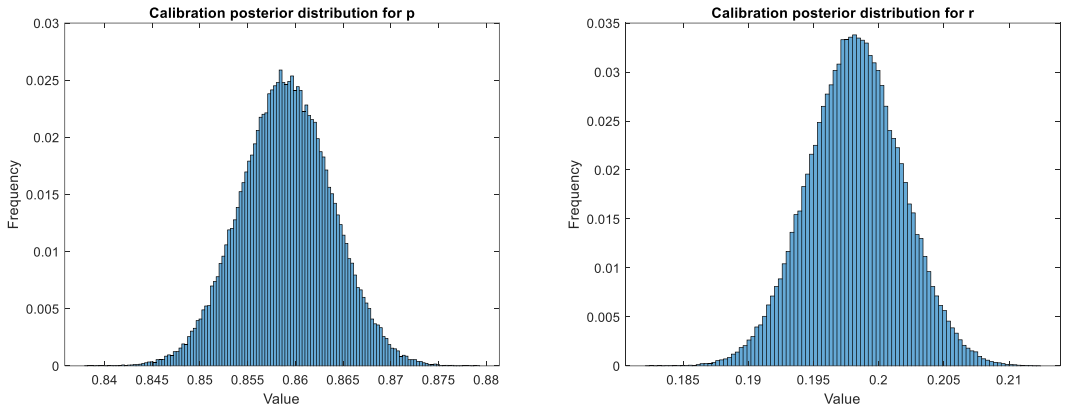


Fig. 1: The calibration posterior distribution for Model One's parameters *p* (left) and *r* (left). The distributions were assumed to be normal with $\sigma_p = 0.005$, $\mu_p = 0.86$, $\sigma_r = 0.004$, and μ_r =0.19

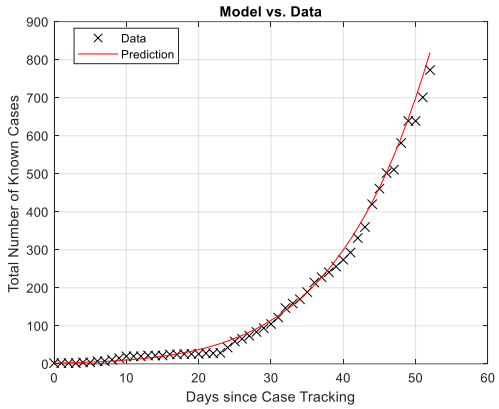


Fig. 2: The calibration data shown with the model with $p = \mu_p$ and $r = \mu_r$.

III. Validation

Next, we wish to update our model to incorporate the full range of data available as well as validate the final model. During this validation step, we similarly use the MH algorithm but our validation prior is the calibration posterior, i.e.,

$$\pi(\boldsymbol{\theta}|\boldsymbol{y_v}, \boldsymbol{y_c}, S_v, S_c) \propto \pi(\boldsymbol{y_v}|\boldsymbol{\theta}, \boldsymbol{y_c}, S_v, S_c)\, \pi(\boldsymbol{\theta}|\boldsymbol{y_c}, S_c),$$

or

$$\pi(\boldsymbol{\theta}|\boldsymbol{y_v}, \boldsymbol{y_c}, S_v, S_c) \propto \pi(\boldsymbol{y_v}|\boldsymbol{\theta}, \boldsymbol{y_c}, S_v, S_c)\, \pi(\boldsymbol{y_c}|\boldsymbol{\theta}, S_c)\, \pi(\boldsymbol{\theta}).$$

Upon taking the log of the validation posterior, we see that

$\log\big(\pi(\boldsymbol{\theta}|\boldsymbol{y_v}, \boldsymbol{y_c}, S_v, S_c)\big) \propto -\frac{\|\boldsymbol{y_v} - C(\boldsymbol{\theta})\|_2{}^2}{2\sigma^2} - \frac{\|\boldsymbol{y_c} - C(\boldsymbol{\theta})\|_2{}^2}{2\sigma^2} + \log(\mathit{Uniform}(\boldsymbol{\theta}))$. Now, we update the MH algorithm and again run for one thousand iterations. Figure 3 shows the posterior distributions for *r* and *p*.

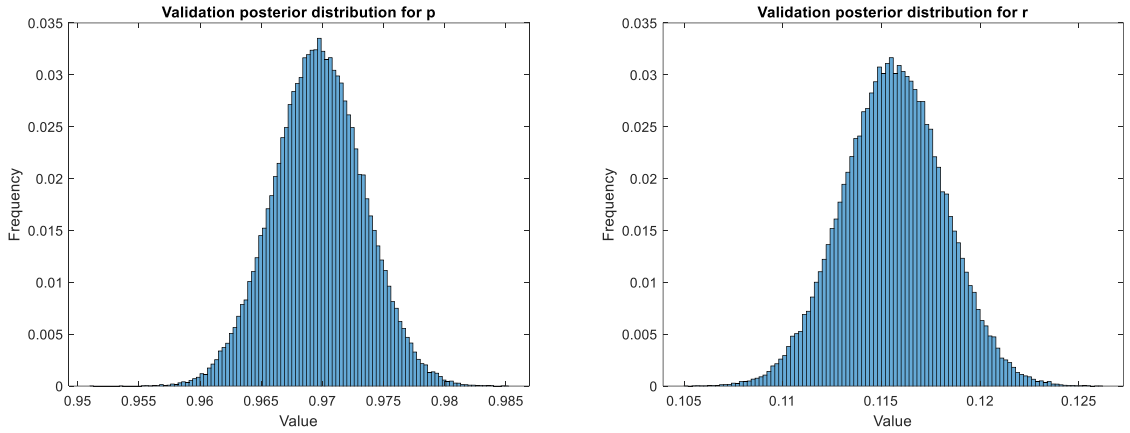


Fig. 3: The validation posterior distribution for Model One's parameters *p* (left) and *r* (left). The distributions were assumed to be normal with $\sigma_p = 0.004, \mu_p = 0.96, \sigma_r = 0.003$, and μ_r =0.12

Using the computed means for *r* and *p* (μ_r and μ_p respectively), Figure 4 shows the prediction compared to the validation data. Viewing the validation data and model in log-scale enables us to see that the growth in the total number of cases changes abruptly around the 60-day mark, and that our model is following that trend rather than previous trends in the data. Given that our QoI is predicting the total number of cases at 100 days, we should only use the data from the 60-day mark forward to validate our model. Then only comparing our model to the total cases between days 60-84 reveals that we are on average 10% off from the measured data.

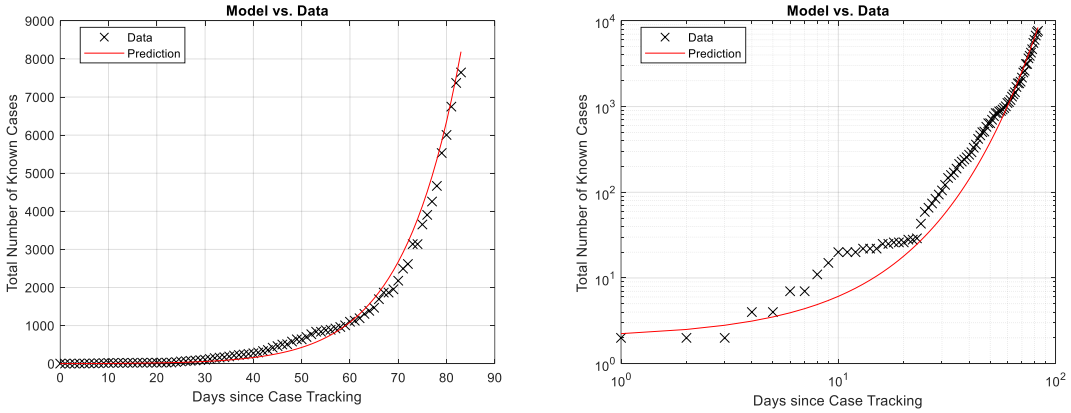


Fig. 4: The validation data shown with the model with $p = \mu_p$ and $r = \mu_r$ (left) and the validation data shown with the model in log-scale.

IV. QoI prediction and uncertainty quantification

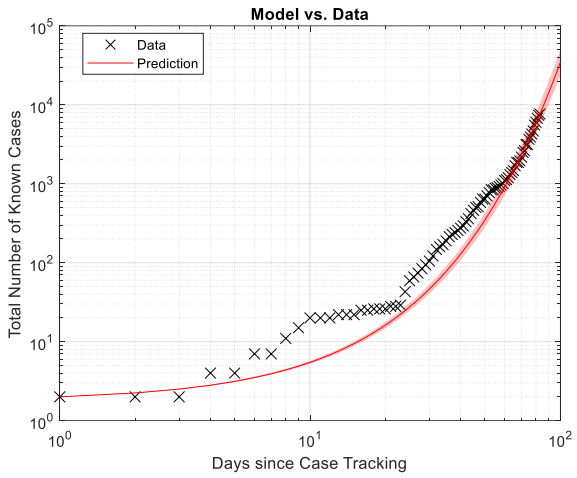


Fig. 5: The validation data shown with the model along with the model's two-sigma uncertainty (shaded region around the prediction)

Our model predicts that there will be 32,757 total cases in Japan after 100 days with a 95% (two-sigma) confidence interval that the number of cases will be between 23,400 and 49,600 . Figure 5 shows the validation data along with our model with accompanying two-sigma uncertainty up to 100 days in log-scale.

Chapter 7

ASSIGNMENT No. 4 **Jaime D. Mora Paz**
CSE 397 (61090), Spring 2020 EID: jdm4335
Prof. J. Tinsley Oden Date: 05/01/2020

Problem on Bayesian Inversion

Following the numbering given in the second document for this assignment, we are working with models 1 and 3.

Model 1 is given by

$$\hat{C}(t) = d_1(t,\theta) = \begin{cases} \left(r(1-p)t + C_0^{1-p}\right)^{1/(1-p)} & 0 \leq p < 1, \\ C_0 \exp(rt) & p = 1, \end{cases} \tag{1}$$

where $r > 0$ and $p \in [0,1]$ are the model parameters ($\theta = (r,p)$). $C_0 = C(0)$ is the initial condition which is going to be set to $C_0 = 2$.

Model 3 is given by

$$\hat{C}(t) = d_3(t,\theta) = a \exp\left[b\left(1 - \frac{1}{1-(1-t/T)^p}\right)\right] + C_0, \tag{2}$$

$$= \exp\left[\ln a + b\left(1 - \frac{1}{1-(1-t/T)^p}\right)\right] + C_0, \tag{3}$$

where $a, b > 0$ are to be determined, while $T = 100$, $C_0 = 20$ and $p = 4$ are fixed a priori. For reasons that will turn clear below, we don't directly regard a as a parameter but $\ln a$ instead. Then our model parameters in this case are $\theta = (\ln a, b)$

We have been given data of confirmed cases of COVID-19 in three countries: United States ($i = 1$), South Korea ($i = 2$) and Japan ($i = 3$). The data covers $N = 84$ days (from $t = 0$ through $t = 83$). The goal is to propose a prediction for $t = 100$ using a calibrated and, if possible, validated model from the ones above.

As suggested, the first $N_C = 50$ numbers in each time series are serving as the calibration data, whereas the remaining $N_V = 34$ will be the validation data.

The quantity of interest is the number of confirmed cases at certain time t_q, $C(t_q)$.

1 General assumptions and definitions

We assume that the relationship between the data and the model output corresponds to a multiplicative noise model, that is,

$$C(t) = d_M(t,\theta)\epsilon, \tag{4}$$

where $\epsilon \sim \text{Lognormal}(0, \sigma_\epsilon^2)$ is a random variable whose logarithm is a Gaussian of mean zero and variance σ_ϵ^2, and M represents the model number.

In our computations, the (logarithmic) noise standard variation is equal for all models and countries, setting it to $\sigma_\epsilon = 0.5$.

The current problem deals with growth along time, with a range of several orders of magnitude. Switching to a logarithmic scale, both in the data and the model, seems natural for dealing with this kind of growth. By taking the natural logarithm of (4) we have

$$\ln C(t) = \ln d_M(t, \theta) + \ln \epsilon.$$

In this logarithmic form, the noise has become additive. The assumption on the noise distribution together with (1) imply that

$$\pi(\ln C(t)|\theta) \propto \mathcal{N}(\ln C(t)| \ln d_M(t, \theta), \sigma_\epsilon^2),$$

As a likelihood for model M, scenario S and the n-th data pair of the data set $Y^S = \{(t_n^S, \ln C_n^S),\ n = 1, 2, ..., N_S\}$, the previous density function is reinterpreted as a function over θ by

$$\pi_{like,M}^n(\theta|S, Y^S) \propto \mathcal{N}(\ln C_n^S| \ln d_M(t_n^S, \theta), \sigma_\epsilon^2), \tag{5}$$

giving rise, in turn, to the joint likelihood distribution

$$\pi_{like,M}(\theta|S, Y^S) = \prod_{n=1}^{N_S} \pi_{like,M}^n(\theta|S, Y^S). \tag{6}$$

The strategy to attempt a predicted value of C follows the prediction pyramid taught in class: calibration, validation and (if model not invalid) prediction of a quantity of interest. We consider the calibration and validation two different scenarios that are introduced to the same Bayesian inversion procedure that we now describe.

Provided a model M (1 or 3), a scenario S ($S = S_{C,i}$ for calibration or $S = S_{V,i}$ for validation, with respect to country i), data Y^S and the corresponding prior pdf $\pi_{prior,M}(\theta|S)$ and likelihood distribution $\pi_{like,M}(\theta|S, Y^S)$, by Bayes' rule the posterior pdf is proportional to the product:

$$\pi_{post,M}(\theta|S, Y^S) \propto \pi_{like,M}(\theta|S, Y^S)\pi_{prior,M}(\theta|S). \tag{7}$$

In this work we will not compute the evidence, so $\pi_{post,M}$ will not be necessarily a normalized pdf.

Inspired by the example on Bayesian linear regression presented by Bishop [1, pp.152-156], we want to obtain the posterior distribution by sequentially getting the individual contribution of each data point, allowing for a so called *on-line or sequential learning*. Let $1 \leq n \leq N_S$, then define the posterior at each step by:

$$\pi_{post,M}^n(\theta|S, Y^S) \propto \begin{cases} \pi_{like,M}^1(\theta|S, Y^S)\pi_{prior,M}(\theta|S) & \text{if } n = 1, \\ \pi_{like,M}^n(\theta|S, Y^S)\pi_{post,M}^{n-1}(\theta|S, Y^S) & \text{if } 2 \leq n \leq N_S, \end{cases} \tag{8}$$

with N_S being either N_C or N_V, according to the current stage. By (6) it is clear that

$$\pi_{post,M}(\theta|S, Y^S) = \pi^{N_S}_{post,M}(\theta|S, Y^S).$$

Thus, we can also have our own version of his Figure 3.7, where the prior, likelihood and posterior distributions at each step are plotted. This will be expanded in the results section.

Next, we choose the prior distributions for the two models, followed by presenting the components of the Bayesian inversion process used at each stage.

2 Prior distributions

2.1 Model 1

The first parameter $\theta_1 = r$ is a positive real. Since it must not be zero and the probability of being large must decay, we propose a log-normal distribution

$$\pi^r_{prior}(\theta_1) = \text{Lognormal}(\theta_1|0, 1).$$

The second parameter $\theta_2 = p$ is a real number in the closed unit interval. Since there is no additional prior information, we set a uniform distribution

$$\pi^p_{prior}(\theta_2) = \mathcal{U}(\theta_2|0, 1).$$

We assume independence of events for the parameters, and then the joint probability is

$$\pi^{r,p}_{prior}(\theta) = \pi^r_{prior}(\theta_1)\pi^p_{prior}(\theta_2). \tag{9}$$

2.2 Model 3

In this model, the number of cases starts at an initial value C_0 and when time reaches $t = T$, $\hat{C}$ converges to a steady state with number of cases equal to $a + C_0$. The value of a might be any number between 1 and the total number of susceptible people in the country, which we may say is the whole population. To reduce the magnitude of the variations of a we may take its logarithm as the actual parameter, as mentioned above. Notice that, disregarding the effect of C_0, our logarithmic model 3 is linear with respect to our chosen parameter $\theta_1 = \ln a$.

Let P_i denote the population of country i, then our prior for $\ln a$ is the uniform pdf:

$$\pi^{\ln a}_{prior}(\theta_1) = \mathcal{U}(\theta_1|0, \ln P_i).$$

Concerning b, it must be positive and should have a decaying pdf, just as in the model 1:

$$\pi^{b}_{prior}(\theta_2) = \text{Lognormal}(\theta_2|0,1).$$

As a consequence, the joint prior probability for this model is

$$\pi^{\ln a,b}_{prior}(\theta) = \pi^{\ln a}_{prior}(\theta_1)\pi^{b}_{prior}(\theta_2). \tag{10}$$

3 Calibration stage

The calibration prior for model 1 is given by (9):

$$\pi_{prior,1}(\theta|S_{C,i}) = \pi^{r,p}_{prior}(\theta).$$

The calibration prior for model 3 is given by (10):

$$\pi_{prior,3}(\theta|S_{C,i}) = \pi^{\ln a,b}_{prior}(\theta).$$

Both priors are independent of the country number i.

We are provided the calibration data sets $Y^{S_{C,i}}$, each consisting of N_C pairs.

The likelihood is determined by function (5), where it is required the evaluation of $d_M(t_n^{S_{C,i}}, \theta)$ for a fixed $t = t_n^{S_{C,i}}$ and varying θ, either through (1) or (3) according to the model number M. This must be done for all $1 \leq n \leq N_C$.

After applying (7) or (8), we obtain the posterior $\pi_{post,M}(\theta|S_{C,i}, Y^{S_{C,i}})$.

4 Validation stage

The prior for the validation stage of country i is set to the respective calibration posterior:

$$\pi_{prior,M}(\theta|S_{V,i}) = \pi_{post,M}(\theta|S_{C,i}, Y^{S_{C,i}}).$$

Identically as above, using the validation data sets $Y^{S_{V,i}}$, each with N_V pairs, we can compute the likelihood $\pi^{n}_{like,M}(\theta|S_{V,i}, Y^{S_{V,i}})$ for every $1 \leq n \leq N_V$.

This is followed by the determination of the validation posterior $\pi_{post,M}(\theta|S_{V,i}, Y^{S_{V,i}}, S_{C,i}, Y^{S_{C,i}})$ using again (7) or (8).

We must choose a metric and a tolerance to decide whether the model is invalid or not. In order to carry this out, we want to have a realization of the model using the mean of the posterior distribution, or a good approximation to it.

By using a MCMC sampling method, we may compute $\bar{\theta}_{M,i}$, the arithmetic mean of all the points in the sample, and use those parameter values to evaluate (1) and (3) at the desired value of t. With this in mind the metric is

$$\delta\left(d_M(\cdot,\bar{\theta}_{M,i}),(Y^{S_{C,i}}\cup Y^{S_{V,i}})\right)=\left(\frac{\sum_{n=1}^{N}|\ln d_M(t_{n,i},\bar{\theta}_{M,i})-\ln C_{n,i}|^2}{\sum_{n=1}^{N}|\ln C_{n,i}|^2}\right)^{1/2}, \tag{11}$$

where

$$(t_{n,i},C_{n,i})=\begin{cases}(t_n^{S_{C,i}},C_n^{S_{C,i}}) & \text{if } 1\le n\le N_C,\\ (t_{n-N_C}^{S_{V,i}},C_{n-N_C}^{S_{V,i}}) & \text{if } N_C+1\le n\le N_C+N_V.\end{cases}$$

Under this metric, which is simply a relative ℓ^2 error in terms of logarithmic values, the tolerance is set to $\gamma=0.05$.

If for any model M and data of country i we get $\delta\le\gamma$, we say that $d_M(\cdot,\bar{\theta}_{M,i})$ is *not invalid*. If so, we can use it to evaluate at $t_q=100$ and propose a predicted value.

The results obtained through this metric must be seen cautiously though. The idea behind using the logarithmic values helps balancing the differences in orders of magnitude of C as time varies. But it may happen that the error is concentrated in the last portion of the curve, where small differences in the log scale correspond to pretty large differences in the regular linear scale.

5 Results

5.1 Implementation

The problem solution was addressed by means of a MATLAB code, using many of the functionalities of that program. We have taken advantage of the predefined probability density functions of MATLAB, as well as the sampling, graphics, function handles and capacity of dealing with floating point numbers of impressively small magnitude. Because of the latter, and that the problem complexity was not very high, we have been able of doing computations that in other circumstances would become intractable. The models and priors are written in different MATLAB files. Later, inside the program driver we use those routines through function handles (objects of the form `y = @(x) f(x)`). Consequently, our posterior distributions can be obtained through recursive multiplications of function handles. Thus, we actually have access to the exact function determining the posterior distribution and we can sample from it, or directly evaluate it and plot it.

As we mentioned initially, we want to visualize the sequential Bayesian learning process. For that purpose, let us start by describing the calibration stage with the help of Figure 1 (all figures have been moved to the end of the document). Here, for one model and one country, the prior distribution is first shown. In the next row, we have the likelihood of the parameters in regard

of the first calibration data point. This likelihood function is scaled so that its peak has a height close to 1. In the right column we then find the posterior for this first data point, which is the multiplication of the previous two functions. In the next row, we have the likelihood corresponding to the second calibration data point. The second posterior is therefore the multiplication of the previous two functions. This process is continued for $n = 3$, as observed in the bottom row. After all data points are exhausted, we obtain the last likelihood and last posterior of the calibration stage, as depicted on Figure 2. This plot allows perceiving a shape of the likelihood quite different from the initial steps, what is caused by the larger values of the data.

In our code, the function handle of the calibration posterior passes directly to serving as the prior for the validation stage, save for a re-scaling. After doing so, we proceed with the exact same procedure as in the calibration, until all the validation data points are exhausted. On Figure 3 we show the validation prior, which is clearly just a stretched version of the posterior of the previous figure. In the bottom, we find the likelihood and posterior of the last step of the validation stage. Notice how the shape of the validation posterior changed with respect to that of the calibration.

The resulting posterior distribution is used inside the MCMC sampling. For this task, we define a subset of the parameter space where we focus our attention (also used for plotting). We use a rectangle $[0, L_x] \times [0, L_y]$. For model 1, $L_x = 2$ and $L_y = 1$. For model 3, $L_x = \ln P_i$ (it depends on the country) and $L_y = 100$. The sampling is done with the `mhsample` function. The starting point of the Markov chain is a parameter vector equal to $(L_x/2, L_y/2)$. We decide the number of samples to be 10,000, which are obtained after a burn-in of 1,000. The randon number generator for the *symmetric* proposal distribution is a bivariate Gaussian of covariance equal to $\begin{pmatrix} L_x/20 & 0 \\ 0 & L_y/20 \end{pmatrix}$.

To visualize the results of the sampling process, we use a bivariate histogram and a 3D scatter plot of $\pi_{post}(\theta_1, \theta_2)$ evaluated at every sample point. In Figure 4 we have one posterior distribution followed by the histogram and the scatter plot with the 10,000 points. Notice that both plots are well localized on the area where the peak of the posterior is. An arithmetic mean of this sample is seemingly a good approximation to the actual distribution's mean. As mentioned above, we take that mean to generate a realization of the calibrated/validated model. Since this realization is in principle not reproducible, for this exercise we run the code and compute the mean 3 times for each model-country pair, and check the validation metric every time (results reported below).

Finally, when running the code, besides generating the plots, it delivers the value of $\bar{\theta}_{M,i}$ according to the random samples, before and after validation. It also evaluates the error metric and prints whether the model is invalid or not. In the end, the code shows the predicted number of cases in country i using the updated $\bar{\theta}_{M,i}$ in the model M, for $t = t_q$.

The MATLAB scripts are given as an attachment to this report. The code that we provide is capable of plotting all this sequence, to dynamically visualize the sequential Bayesian learning process for any given model and country included in this work. The users can choose the model, country and whether they wish to generate all the plots and save them as an animated GIF.

Now that we have explained the methodology and how the code works, we present the results.

5.2 Model 1

Figure 5 presents the comparison of the data against the results of (one realization of) model 1, in logarithmic terms, before and after the validation process. Moreover, on Table 1 we are reporting the results of three realizations of the model for each country. The information includes the mean $\bar{\theta}_{M,i}$ of samples before and after validation. Next, we show the result of δ obtained with the latter. We compare against the tolerance $\gamma = 0.05$ and, in the sixth column we determine whether the model realization qualifies as invalid or not invalid. Finally, we compute the output of the model for $t_q = 100$ in all cases, indicating those where the model did not fail the validation test. In this case, that happened for one of the countries only (Japan).

Country	**Realization No.**	**Calibration sample mean**	**Validation sample mean**	δ	**Qualifies as**	**Prediction at $t_q = 100$**
U.S.A.	1	$r = 0.1026$, $p = 0.9840$	$r = 0.1466$, $p = 0.9987$	0.1655	INVALID	3978154
U.S.A.	2	$r = 0.1019$, $p = 0.9836$	$r = 0.1540$, $p = 0.9916$	0.1718	INVALID	3626853
U.S.A.	3	$r = 0.0980$, $p = 0.9918$	$r = 0.1494$, $p = 0.9925$	0.1716	INVALID	2629687
South Korea	1	$r = 0.1750$, $p = 0.9904$	$r = 0.2317$, $p = 0.8898$	0.1345	INVALID	121144
South Korea	2	$r = 0.1703$, $p = 0.9899$	$r = 0.2418$, $p = 0.8824$	0.1338	INVALID	113096
South Korea	3	$r = 0.1715$, $p = 0.9885$	$r = 0.2288$, $p = 0.8897$	0.1353	INVALID	111153
Japan	1	$r = 0.1880$, $p = 0.8660$	$r = 0.1874$, $p = 0.8630$	0.0394	NOT INVALID	<u>13158</u>
Japan	2	$r = 0.1958$, $p = 0.8558$	$r = 0.1842$, $p = 0.8675$	0.0389	NOT INVALID	<u>13836</u>
Japan	3	$r = 0.1935$, $p = 0.8564$	$r = 0.1841$, $p = 0.8707$	0.0400	NOT INVALID	<u>15256</u>

Table 1: Results with model 1. Predicted values obtained with not invalid models are underlined.

5.3 Model 3

With the same format as in the first model, we show the data and modeled curves in Figure 6, while the remaining information is given on Table 2. Unlike the previous one, this model turned out to be invalid for every country.

Country	**Realization No.**	**Calibration sample mean**	**Validation sample mean**	δ	**Qualifies as**	**Prediction at $t_q = 100$**
U.S.A.	1	$\ln a = 8.6132$, $b = 28.9367$	$\ln a = 12.6497$, $b = 76.3253$	0.1188	INVALID	311685
U.S.A.	2	$\ln a = 8.5315$, $b = 28.2734$	$\ln a = 12.6631$, $b = 76.6859$	0.1189	INVALID	315897
U.S.A.	3	$\ln a = 8.5384$, $b = 28.3272$	$\ln a = 12.6508$, $b = 76.3626$	0.1188	INVALID	312034
South Korea	1	$\ln a = 10.7577$, $b = 19.1488$	$\ln a = 9.5783$, $b = 14.1601$	0.1157	INVALID	14468
South Korea	2	$\ln a = 10.7661$, $b = 19.1412$	$\ln a = 9.5744$, $b = 14.1643$	0.1158	INVALID	14411
South Korea	3	$\ln a = 10.7715$, $b = 19.1754$	$\ln a = 9.5705$, $b = 14.1330$	0.1157	INVALID	14356
Japan	1	$\ln a = 6.6987$, $b = 7.5260$	$\ln a = 7.6714$, $b = 12.2328$	0.1255	INVALID	2166
Japan	2	$\ln a = 6.7085$, $b = 7.6011$	$\ln a = 7.6616$, $b = 12.1813$	0.1256	INVALID	2145
Japan	3	$\ln a = 6.6948$, $b = 7.5713$	$\ln a = 7.6715$, $b = 12.2980$	0.1256	INVALID	2166

Table 2: Results with model 3. All the models are qualified as invalid with the preset tolerance.

6 Conclusions

The Bayesian sequential learning has been applied to the modeling of confirmed cases of COVID-19. Because of the quick change in order of magnitude in the data, we switched to logarithmic scale, in which a single noise model was sufficient to work for all cases.

Two different simple models were subjected to the process. By setting constant initial conditions, we have reduced the parameter space of both models to two variables only. This furthermore helps visualizing the evolution of the learning process.

Only one model, applied to only one country, passed the validation test and was qualified as *not invalid*. From Table 1 we see that we are talking about model 1 and Japan, with a predicted number of cases for $t_q = 100$ of around 13158, 13836 or 15256.

A more complete work may comprise the calibration of a larger parameter space, including C_0 and the hyperparameter σ_ϵ. The same strategy taken in this work would still apply, but the visualization of the Bayesian sequential learning would be less straightforward.

A MATLAB code and animated GIFs with the results herein presented are attached to this report.

References

[1] Bishop, Christopher M., *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] Oden, J. Tinsley. *Foundations of Predictive Computational Science*. Lecture notes. The University of Texas at Austin, 2017.

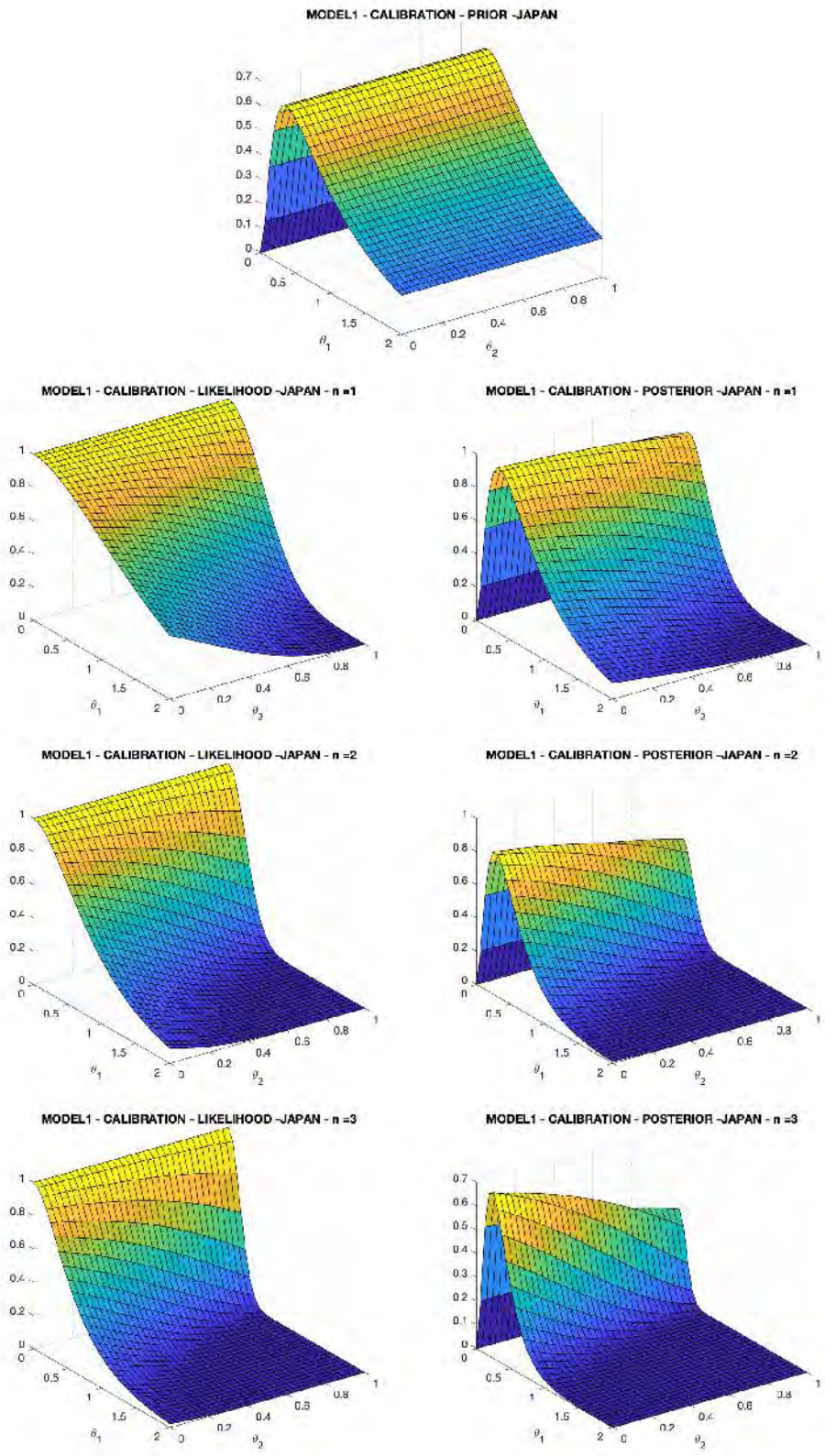


Figure 1: Bayesian sequential learning - First calibration steps (Model 1, Country 3).

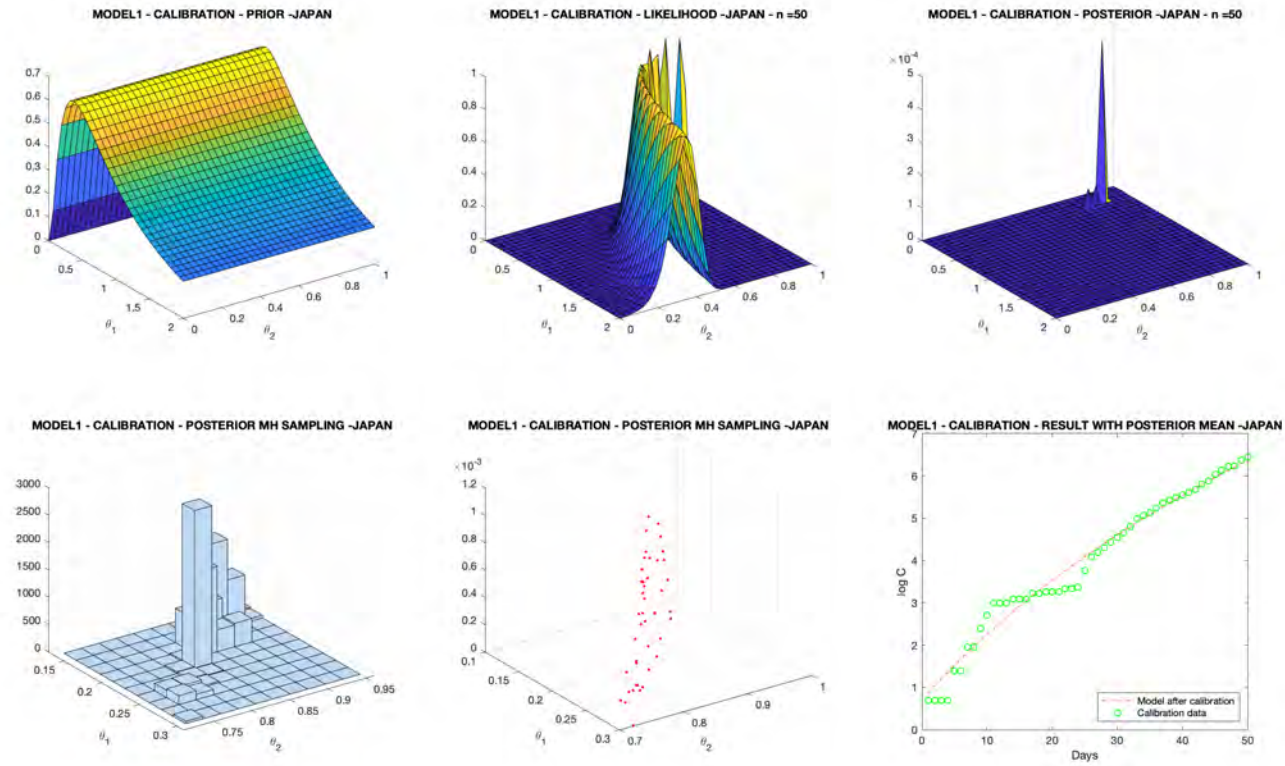


Figure 2: Bayesian sequential learning - Last calibration step (Model 1, Country 3).

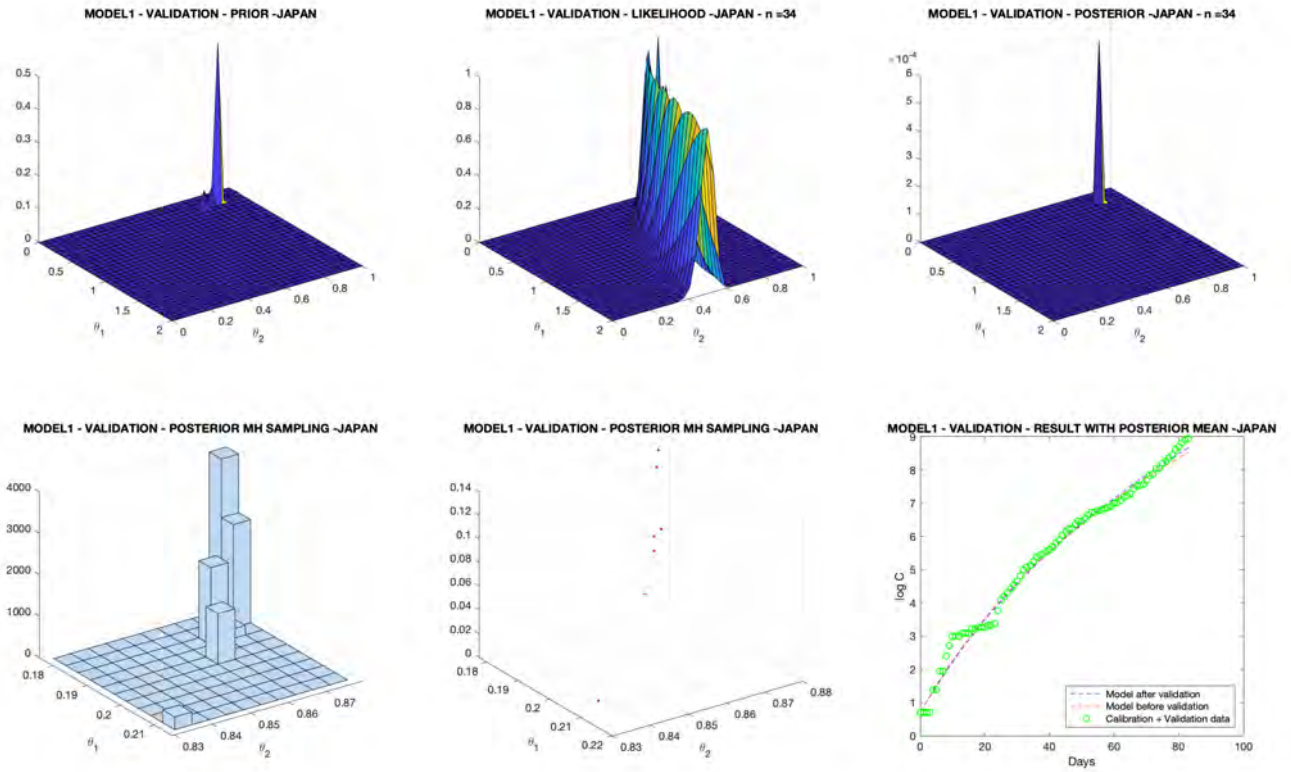


Figure 3: Bayesian sequential learning - Validation stage: prior, and last likelihood and posterior (Model 1, Country 3).

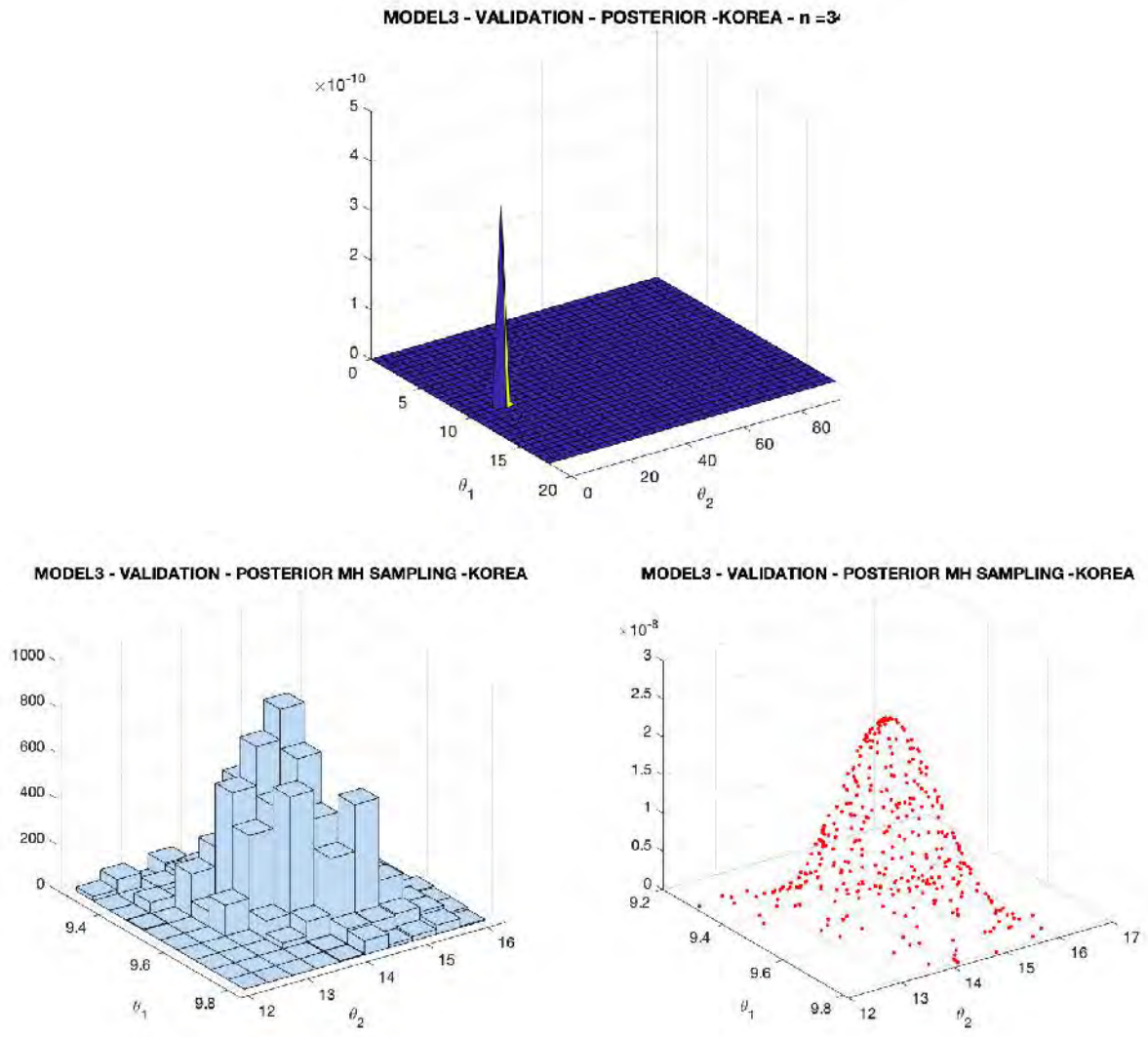


Figure 4: Posterior distribution after validation stage, and sampling through the Metropolis-Hastings MCMC method (Model 3, Country 2).

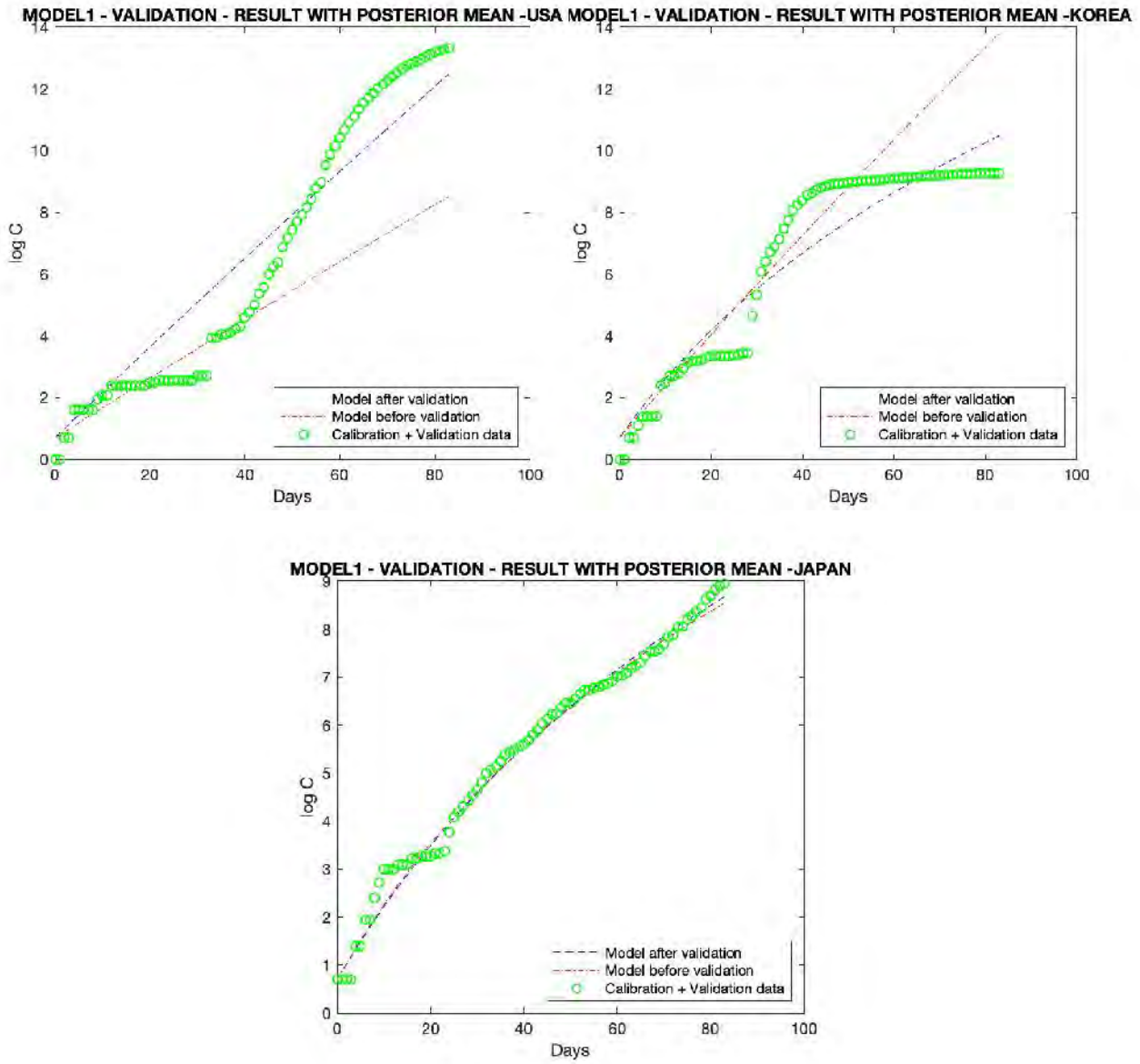


Figure 5: Comparison of data against model 1 before and after the validation stage for each country.

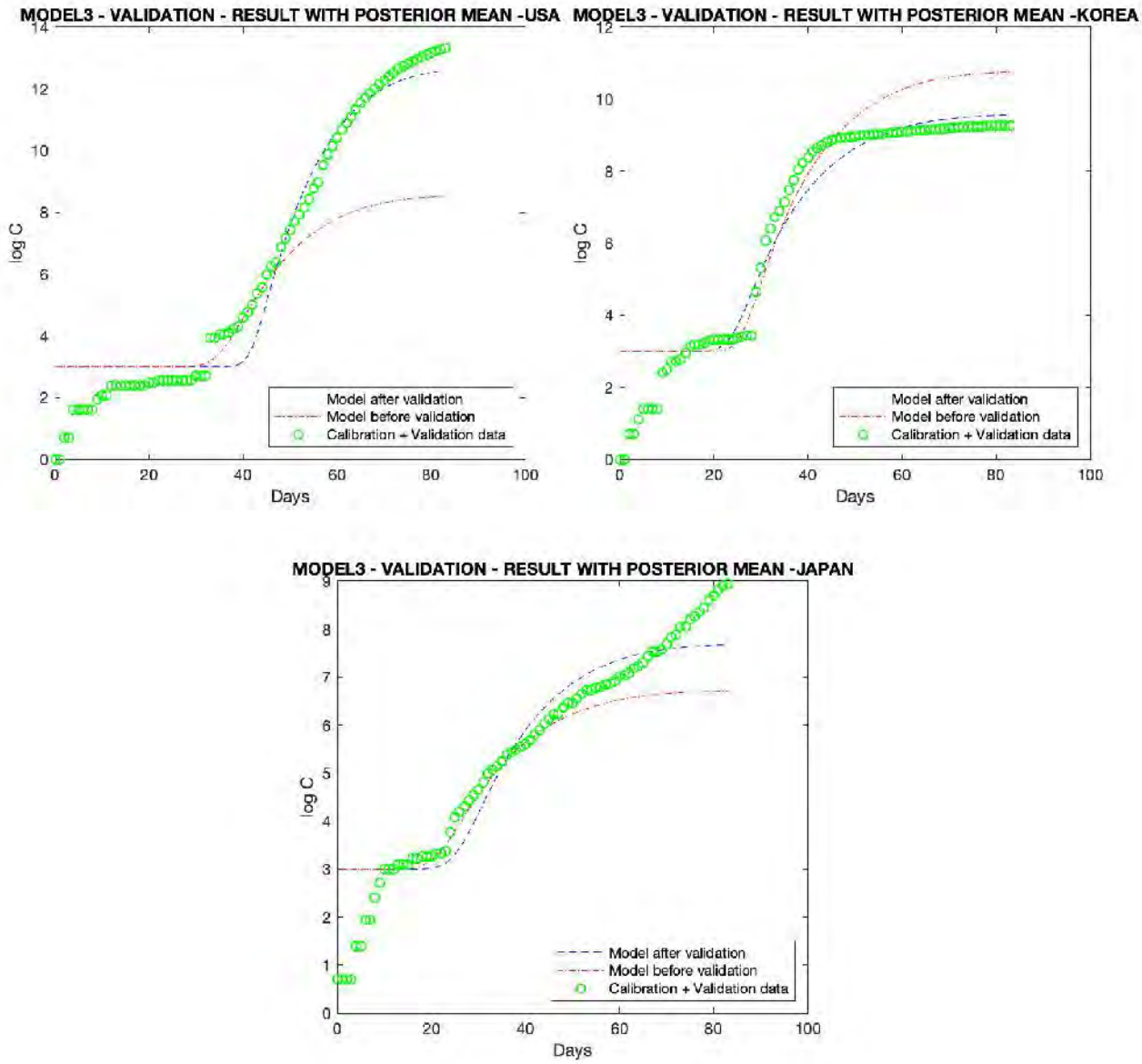


Figure 6: Comparison of data against model 3 before and after the validation stage for each country.

Chapter 8

Bayesian Calibration and Validation of a **COVID**-19 Contagion Model

Cyrus Neary
The University of Texas at Austin

April, 2020

Given data on the number of confirmed Covid-19 cases in each of the 84 days since the first reported case, we wish to model the spread of the virus and to subsequently predict the num-ber of confirmed cases at day $T = 100$. The primary objective of this exercise is to apply the model calibration and validation techniques learned in *CSE 397: Predictive Computational Science Fundamentals* to a real-world model and dataset.

1 The Generalized Growth Model

We use the generalized growth model [1] to predict the number of confirmed cases of COVID-19 at time t. Let $C(t)$ represent the number of confirmed cases after t days have elapsed since the first confirmed case. We define two parameters: the growth rate $r \geq 0$ and the deceleration parameter $p \in [0, 1]$. The generalized growth model is then given by Equation 1.

$$\frac{dC(t)}{dt} = rC(t)^p \tag{1}$$

Let $m = \frac{1}{1-p}$ and $C_0 = C(0)$, then we may write the solution to the above ODE as:

$$C(t) = \begin{cases} rt + C_0 & \text{if } p = 0 \\ (\frac{r}{m}t + C_0^{1/m})^m & \text{if } 0 < p < 1 \\ C_0 e^{rt} & \text{if } p = 1. \end{cases} \tag{2}$$

2 Bayesian Model Calibration and Validation

To express our modeling and prediction goals in the language of Bayesian model calibration, we first define model vector $\theta = (r, p) \in \mathbb{R}^2$. Our aim to find the posterior distribution $\pi(\theta|\mathbf{y})$: the density function of parameter vector θ given the observed data $\mathbf{y} = \{y_0, y_1, ..., y_{83}\}$. While we do not know this posterior distribution, we may compute it using Bayes' theorem.

$$\pi(\theta|\mathbf{y}) = \frac{\pi(\mathbf{y}|\theta)\pi(\theta)}{\pi(\mathbf{y})} \tag{3}$$

Here, $\pi(\mathbf{y}|\theta)$ is the likelihood of data $\mathbf{y}$ being produced by the model with parameters θ. We assume this distribution is generated by our model and additive observation noise. The prior distribution $\pi(\theta)$ encodes our prior knowledge of the values that θ may take. $\pi(\mathbf{y})$ is called the evidence.

Once we have the posterior distribution, we obtain a distribution over the values of our QoI: $\pi(Q(\theta)|\mathbf{y})$. We then predict the QoI to be the average value of this distribution $\mathbb{E}[Q(\theta)|\mathbf{y}]$ and we use the variance of the QoI's distribution to describe the uncertainty in this prediction.

Following the "prediction pyramid" hierarchy discussed in class, we break our process up into three tasks: model calibration, model validation, and prediction of the QoI.

2.1 Model Calibration

In model calibration, we choose a prior $\pi(\theta)$ for the calibration scenario S_c and use preliminary calibration data $\mathbf{y_c}$ to obtain the calibration posterior $\pi(\theta|\mathbf{y_c}, S_c)$. This stage is meant to calibrate our model by finding parameters using data that is typically obtained through simple component tests.

$$\pi(\theta|\mathbf{y_c}, S_c) = \frac{\pi(\mathbf{y_c}|\theta, S_c)\pi(\theta)}{\pi(\mathbf{y_c}|S_c)} \tag{4}$$

2.2 Model Validation

Once the calibration posterior has been obtained, we move to the validation scenario S_v. We typically validate the model using data $\mathbf{y_v}$ generated by experiments specifically designed to test our model's ability to accurately predict the QoI. Mathematically, we use Bayes' rule to update our previous estimate of the posterior distribution on θ to include $\mathbf{y_v}$ as described in Equation 5.

$$\pi(\theta|\mathbf{y_v}, \mathbf{y_c}, S_v, S_c) = \frac{\pi(\mathbf{y_v}|\theta, \mathbf{y_c}, S_v, S_c)\pi(\theta|\mathbf{y_c}, S_c)}{\pi(\mathbf{y_v}|y_c, S_v, S_c)} \tag{5}$$

At this stage, we use a problem-specific metric $d(.,.)$ to verify that our model is sufficiently accurate. To do this, we pre-specify some tolerance γ_{tol}, and check that $d(M(), \mathbf{y_v}) \leq \gamma_{tol}$, where $M(\theta)$ denotes the model output given the selected parameter vector θ. If this inequality is satisfied, we say that our model is valid with respect to the selected metric and tolerance, and we move onto prediction of the QoI using the posterior distribution described by Equation 5.

3 Markov Chain Monte Carlo (MCMC)

Because we aren't able to obtain closed-form expressions for the posterior distributions described in Section 2, we use Markov Chain Monte Carlo as a way to sample them. This technique works by constructing a Markov chain that has the desired posterior distribution as its limiting distribution. We may thus approximate the posterior distribution by simulating random walks through the designed Markov chain. In this project, we use the Metropolis Hastings Algorithm for MCMC.

4 Calibrating a Model for the Japan Dataset

In this project I use data on the confirmed cases in Japan. Because we are not able to design and test experiments to create a separate validation scenario for our model, the existing 84-day dataset is instead split into calibration and validation components. Data from the first 50 days $\mathbf{y_c} = \{y_0, y_1, ..., y_{49}\}$ is used to calibrate the model, while the data from the remaining days $\mathbf{y_v}\{y_{50}, y_{51}, ..., y_{83}\}$ is used to validate it.

For a given parameter vector θ, we may compute the value of the posterior distribution $\pi(\theta|\mathbf{y_c}, S_c)$ using Equation 4. Here, the prior $\pi(\theta)$ is taken to be a uniform distribution on the possible parameter values. To compute the likelihood $\pi(\mathbf{y_c}|\theta, S_c)$, we assume that all observation error is described by additive Gaussian noise with zero mean and standard deviation σ.

For a given data-point, y_t the number of confirmed cases at day t, we may then compute the likelihood $\pi(y_t|\theta, S_c)$ for a selected instantiation of θ using Equation 6.

$$\pi(y_t|\theta, S_c) \propto e^{\frac{(y_t - C_\theta(t))^2}{2\sigma^2}} \tag{6}$$

Assuming the gaussian noise is sampled i.i.d. with each new data-point, the likelihood of the entire calibration dataset may be written as in Equation 7.

$$\pi(\mathbf{y_c}|\theta, S_c) \propto e^{\frac{1}{2\sigma^2}\sum_t (y_t - C_\theta(t))^2} \tag{7}$$

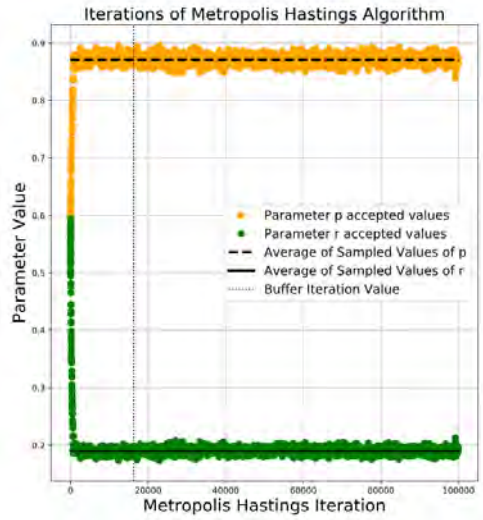
A quantity proportional to the log of the desired posterior distribtuion may then be computed for a particular value of θ using Equation 8.

$$log(\pi(\theta|\mathbf{y_c}, S_c)) \propto log(Uniform(\theta)) - \frac{1}{2\sigma^2}\sum_t (y_t - C_\theta(t))^2 \tag{8}$$

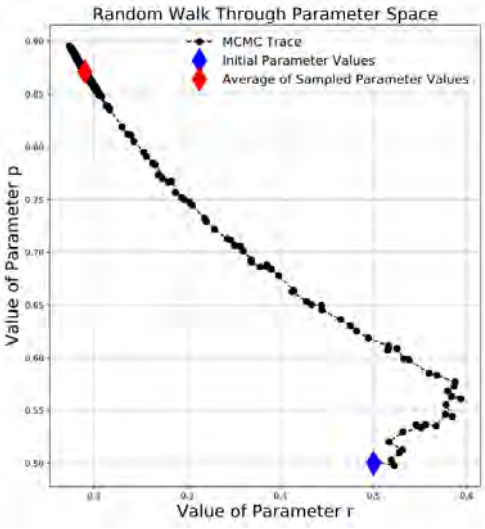
Note that to sample $\pi(\theta|\mathbf{y_c}, S_c)$ we use the Metropolis-Hasings algorithm. To construct the Markov chain for MCMC, this algorithm uses only relative values of the posterior distribution for different points θ, θ' in the parameter space. So, we need not compute the evidence $\pi(\mathbf{y_c}|S_c)$.

In our implementation, we used a value of $\sigma = 8$ to model the gaussian noise.

The results of running MCMC are shown in Figure 1. The random walk through the parameter space quickly converges towards an area in the top left of Figure 1b, where it appears to stay in subsequent iterations of the algorithm. This indicates that it is in this area of the parameter space where the posterior density is the highest. To compute an estimate of the average value of θ under the posterior distribution $\pi(\theta|\mathbf{y_c}, S_c)$, denoted $\hat{\theta}$, we compute the empirical average of the accepted parameter values after a certain number of buffer iterations. We exclude all algorithm-accepted parameters before the specified number of buffer iterations to ensure the random walk has had sufficient time to settle into the equilibrium distribution of the constructed Markov chain.



(a) Parameter values accepted by the Metropolis Hastings algorithm. The parameter values were initialized to $\theta = (0.5, 0.5)$.



(b) The random walk through the parameter space resulting from MCMC.

Figure 1 – A visualization of the process of running the Metropolis-Hastings algorithm for MCMC to sample the posterior distribution $\pi(\theta|\mathbf{y_c}, S_c)$.

Once we have an estimate $\hat{\theta}$ of the mean parameter vector under the posterior distribution $\pi(\theta|\mathbf{y_c}, S_c)$ for the calibration scenario, we may construct the corresponding model solution $C_{\hat{\theta}}(t)$ and plot it as a function of t to verify visually that it fits the calibration data. This plot is shown in Figure 2.

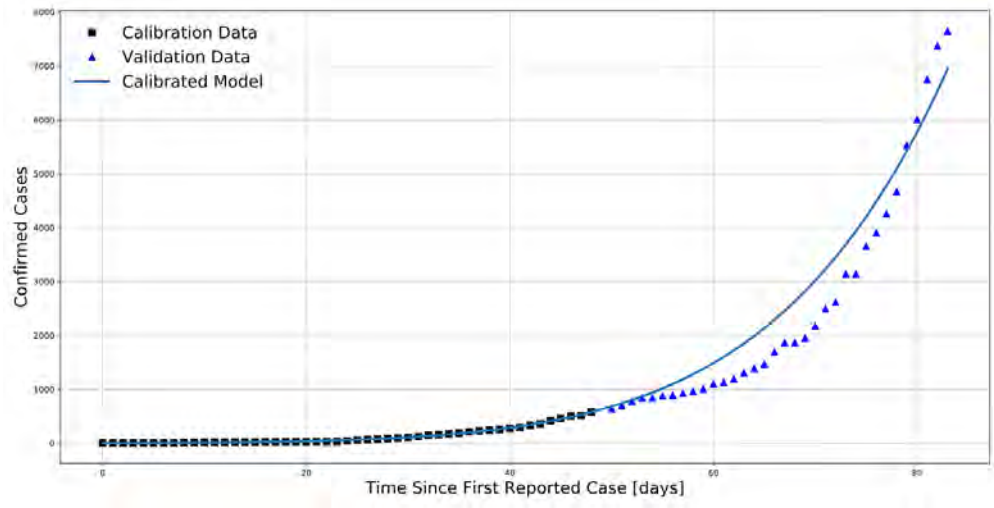


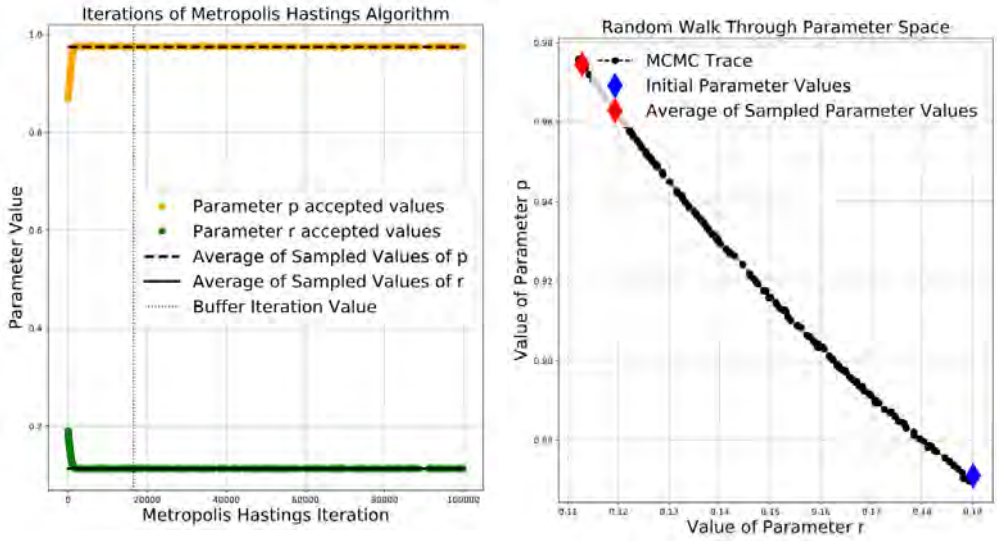
Figure 2 – Confirmed case data and calibrated model predictions. Black rectangles show the confirmed case data-points included in the calibration data-set. The blue curve shows the predicted number of confirmed cases given by the model $C_{\hat{\theta}}(t)$, where $\hat{\theta}$ is our estimate of the mean value of θ computed using our MCMC-sampled parameter values.

5 Model Validation

To validate the model we use MCMC to sample the posterior distribution $\pi(\theta|\mathbf{y_v}, \mathbf{y_c}, \mathbf{S_v}, \mathbf{S_c})$. This distribution is conditioned on the validation data $\mathbf{y_v} = \{y_{50}, y_{51}, ..., y_{83}\}$ as well as the information gained from the calibration step. For any given sample of the parameter vector θ, we use Equation 5 to compute the value of $\pi(\theta|\mathbf{y_v}, \mathbf{y_c}, \mathbf{S_v}, \mathbf{S_c})$. Following similar mathematical steps to those described in Section 4, we obtain the following expression for the log of the posterior distribution.

$$log(\pi(\theta|\mathbf{y_v}, \mathbf{y_c}, S_v, S_c)) \propto log(\pi(\theta|\mathbf{y_c}, S_c))) - \frac{1}{2\sigma^2} \sum_{t \in S_v} (y_t - C_\theta(t))^2 \tag{9}$$

Where $\pi(\theta|\mathbf{y_c}, S_c)$ is estimated to be a multi-variate normal distribution with mean $\boldsymbol{\mu}$ and covariance matrix Σ. The values of $\boldsymbol{\mu}$ and Σ are computed as the empirical mean and covariance of the accepted sample points from the calibration run of MCMC described in Section 4. The results of the Metropolis-Hastings algorithm are shown in Figure 3. For validation, the algorithm is initialized to the mean parameter values computed during calibration. We note from Figure 3 that the inclusion of the validation data $\mathbf{y_v}$ appears to have changed the most likely parameter values.



(a) Parameter values accepted by the Metropolis Hastings algorithm. The parameter values were initialized to $\theta = (0.19, 0.87)$.

(b) The random walk through the parameter space resulting from MCMC.

Figure 3 – A visualization of the process of running the Metropolis-Hastings algorithm for MCMC to sample the posterior distribution $\pi(\theta|\mathbf{y_v}, \mathbf{y_c}, S_c, S_v)$.

Once again, to visually verify that the model fits the data, we compute an empirical estimate $\hat{\theta}$ of the mean value of θ with respect to the posterior $\pi(\theta|\mathbf{y_v}, \mathbf{y_c}, S_v, S_c)$. The resulting model predictions are shown in Figure 4.

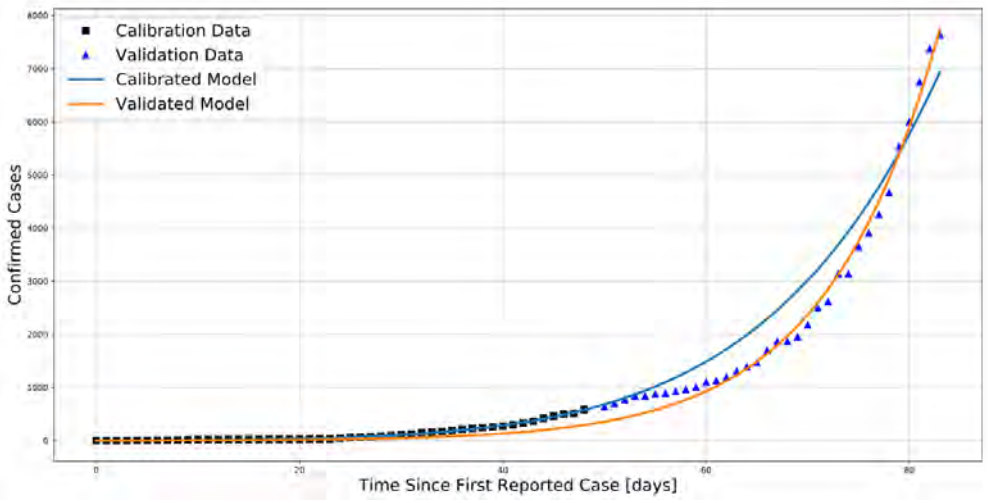


Figure 4 – Confirmed case data and validated model predictions. Blue triangles show the confirmed case data-points included in the validation data-set. The orange curve shows the predicted number of confirmed cases given by the model $C_{\hat{\theta}}(t)$, where $\hat{\theta}$ is our estimate of the mean value of θ computed using our MCMC-sampled parameter values from the validation-scenario posterior distribution.

To finish the validation process, we select the validation metric $d(.,.)$ to be the Chebyshev distance. That is, we wish to verify that $d(C_{\hat{\theta}}(\tau_v), \mathbf{y_v}) = ||C_{\hat{\theta}}(\tau_v) - \mathbf{y_v}||_\infty \leq \gamma_{tol}$. Here, $C_{\hat{\theta}}(\tau_v)$ is a vector representing the model's predictions at the days $\tau_v = \{50, 51, ..., 83\}$ corresponding to the data-points in the validation dataset $\mathbf{y_v}$. As the number of confirmed cases during the time period specified by τ_v are on the order of 1e3, we select $\gamma_{tol} = 500$.

For the model $C_{\hat{\theta}}(t)$ corresponding to the estimate $\hat{\theta}$ of the mean value of θ with respect to distribution $\pi(\theta|\mathbf{y_v}, \mathbf{y_c}, S_v, S_c)$, we observe that

$$||C_{\hat{\theta}}(\tau_v) - \mathbf{y_v}||_\infty = 363 \leq \gamma_{tol}$$

and thus our model is valid with respect to the selected metric and tolerance.

6 Predicting the Quantity of Interest (QoI)

Our QoI, $Q(\theta)$, is the number of confirmed cases at day $T = 100$. In other words, $Q(\theta) = C_\theta(100)$. Given the posterior distribution $\pi(\theta|\mathbf{y_v}, \mathbf{y_c}, S_v, S_c)$ we may compute a probability distribution $\pi(Q(\theta)|\mathbf{y_v}, \mathbf{y_c}, S_v, S_c)$ over the values that Q could take. We would then predict the value of Q to be the mean of this distribution, and use the variance of the distribution to measure our uncertainty in the prediction.

To estimate the mean and variance of Q, we use our samples of $\pi(\theta|\mathbf{y_v}, \mathbf{y_c}, S_v, S_c)$ from Section 5. Let Θ be the set of all MCMC-accepted samples of θ after the number of buffer iterations illustrated in Figure 3a by a dotted line. The value of our prediction can then be computed as:

$$\hat{Q} = \frac{1}{|\Theta|} \sum_{\theta \in \Theta} Q(\theta) = 34587.$$

A measurement of the uncertainty, an estimate of the standard deviation in the value of Q, is computed as:

$$\hat{\sigma}_Q = \sqrt{\frac{(Q(\theta) - \hat{Q})^2}{|\Theta| - 1}} = 113$$

7 References

[1] Cécile Viboud, Lone Simonsen, and Gerardo Chowell. "A generalized-growth model to characterize the early ascending phase of infectious disease outbreaks". In: *Epidemics* 15 (2016), pp. 27–37.

Chapter 9

fit_model1_USA

May 5, 2020

1 Akhil Potla (ap44876)

2 Model 1 US

[1]:
```
import sys
sys.path.insert(0,'../')

import pandas as pd
import numpy as np
import scipy
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

from data import plot_countries_all_plus_normalize_1 as plot
from data import save_country_data_1 as save

# import models
from models import model1, model2, model3

df = pd.read_csv('../../data/datasets/time-series-19-covid-combined.csv')
```

[24]:
```
df_US = pd.read_csv('./US.csv', header=None)
calibration_US, validation_US = df_US.iloc[:50], df_US.iloc[50:]


calibration_times = calibration_US[0].to_numpy()
validation_times = validation_US[0].to_numpy()
```

The following code is based on this tutorial and the starter code for the models provided by Prashant.

Model 1 is imported from this file.

The description for model 1 is from this file.

Model 1 has two parameters that we need to tune for the scenario, r and p. These two parameters together will be represented using θ, where $\theta = (r, p)$

2.1 Noise

We assume Gaussian noise for the model. With an average $\mu = 0$ and a standard deviation σ, which will be selected during calibration and validation by calculating the standard deviation of the samples for their respective steps.

2.2 Prior

The prior for the parameters in θ is considered to be uniform.

- r is nonnegative value.
- p is any value $[0, 1]$

We essure that the value for both r and p is valid. If either of the values is not valid it returns zero, otherwise it returns 1.

2.3 Likelihood

We assume a Gaussian likelihood function.

$$\pi_{like}(y|\theta) \propto exp[-\frac{||Y - C(\theta)||^2}{2\sigma^2}] \tag{1}$$

- $C(\theta)$ is the output of the model with parameter θ.
- $||Y - C(\theta)||^2$ is the squred norm of $|Y - C(\theta)|$

We take the logarithm of the likelihood function because it is more efficient.

$$\log(\pi_{like}(Y|\theta)) \propto -\frac{1}{2\sigma}\sum_{i=1}^{N}|y_i - C(t_i;\theta)|^2 \tag{2}$$

2.4 Posterior

$$\pi_{post}(\theta|Y) = \frac{\pi_{like}(Y|\theta)\pi_{prior}(\theta)}{\pi_{evidence}(Y)} \tag{3}$$

Since the evidence is very difficult to compute, we use MCMC, so we can simplify.

$$\pi_{post}(\theta|Y) \propto \pi_{like}(Y|\theta)\pi_{prior}(\theta) \tag{4}$$

We take the logarithm because it is more efficient.

$$\log(\pi_{post}(\theta|Y)) \propto log(\pi_{like}(Y|\theta)) + log(\pi_{prior}(\theta)) \tag{5}$$

2.5 Calibration

During the calibration step we take the data from the first 50 days and we use it to calibrate the values of the model parameters. The parameters of this model are `r` and `p`. These two parameters together will be represented using θ, where $\theta = (r, p)$

For that we select a transition model to take us from one state to another. The transition model selects the values for `r` and `p` based on the previous value for `r` and `p` and given a reasonable standard deviation. We use a Gaussian/Normal distribution to randomly select the next value for these parameters.

[3]:
```
calibration_cases = calibration_US[1].to_numpy()
calibration_noise = np.std(calibration_cases)
n_iter = 50000
```

[4]:
```
transition_model1 = lambda theta: [np.random.normal(theta[0], 0.1), np.random.
↪normal(theta[1], 0.01)]

def prior(theta):
    if (theta[0] < 1.e-5 or theta[1] <= 1.e-5):
        return 1.e-8
    elif theta[1] >= (1 - 1.e-5):
        return 1.e-8
    else:
        return 1

# likelihood
def log_lik(theta, model, t, data, fix_params, noise_sigma):
    # get model output
    C = model(theta, t, fix_params)
    # log likelihood
    b = data - C
    return (-1. / (2. * noise_sigma * noise_sigma)) * np.linalg.norm(data - C)␣
↪** 2

def acceptance(theta, theta_new):
    if theta_new > theta:
        return True
    else:
        accept = np.random.uniform(0, 1)
        return (accept < (np.exp(theta_new - theta)))

def metropolis_hastings(model, likelihood, prior, transition_model, param_init,␣
↪iterations,
                        t, data, fix_params, acceptance_rule, noise_sigma):
    # likelihood_computer(x,data): returns the likelihood that these parameters␣
↪generated the data
```

```
    # transition_model(x): a function that draws a sample from a symmetric␣
↪distribution and returns it
    # param_init: a starting sample
    # iterations: number of accepted to generated
    # data: the data that we wish to model
    # acceptance_rule(x,x_new): decides whether to accept or reject the new␣
↪sample
    x = param_init
    accepted = []
    rejected = []
    num_accepted = []
    for i in range(iterations):
        x_new = transition_model(x)
        x_lik = likelihood(x, model, t, data, fix_params, noise_sigma)
        x_new_lik = likelihood(x_new, model, t, data, fix_params, noise_sigma)
        if (acceptance_rule(x_lik + np.log(prior(x)),
                            x_new_lik+np.log(prior(x_new)))):
            x = x_new
            accepted.append(x_new)
        else:
            rejected.append(x_new)

    return np.array(accepted), np.array(rejected)
```

We run the Metropolis Hastings Algorithm 50,000 times, and we burn the first 80% of the accepted trials.

[6]:
```
accepted, rejected = metropolis_hastings(model1, log_lik, prior,␣
↪transition_model1, [0., 0.], n_iter,
                                         calibration_times, calibration_cases,␣
↪[1, 100], acceptance, calibration_noise)
print('Number of Accepted: ', len(accepted))
after_burn_in = int(len(accepted) * 0.8)
accepted = accepted.T
```

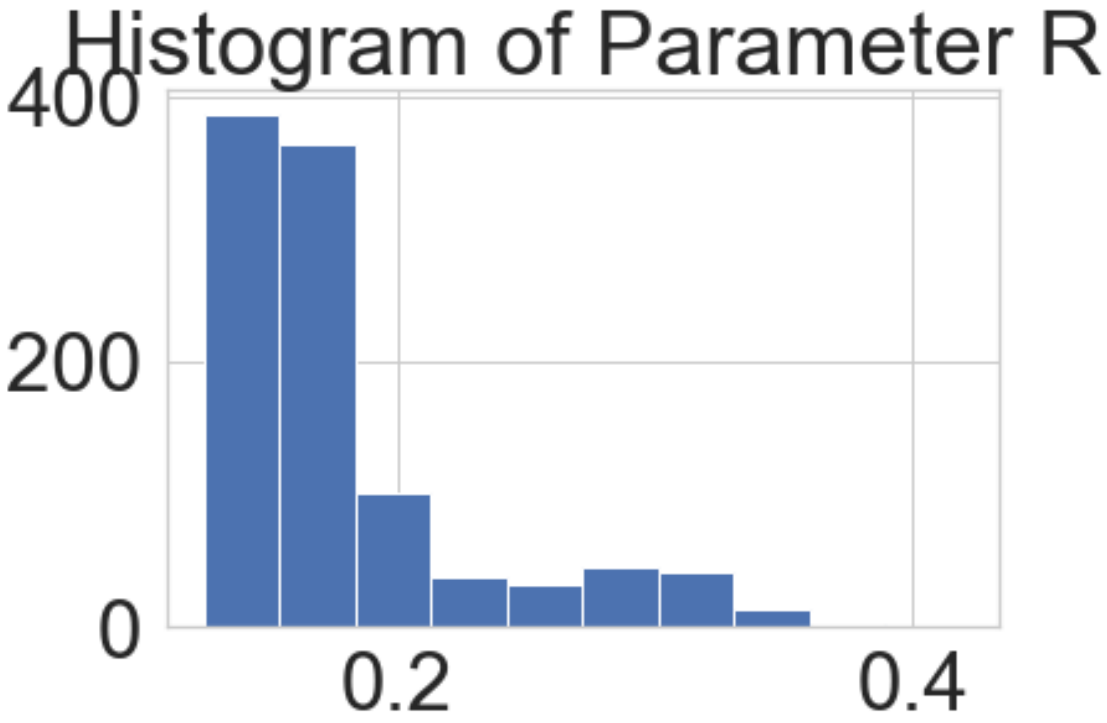
```
Number of Accepted:  5125
```

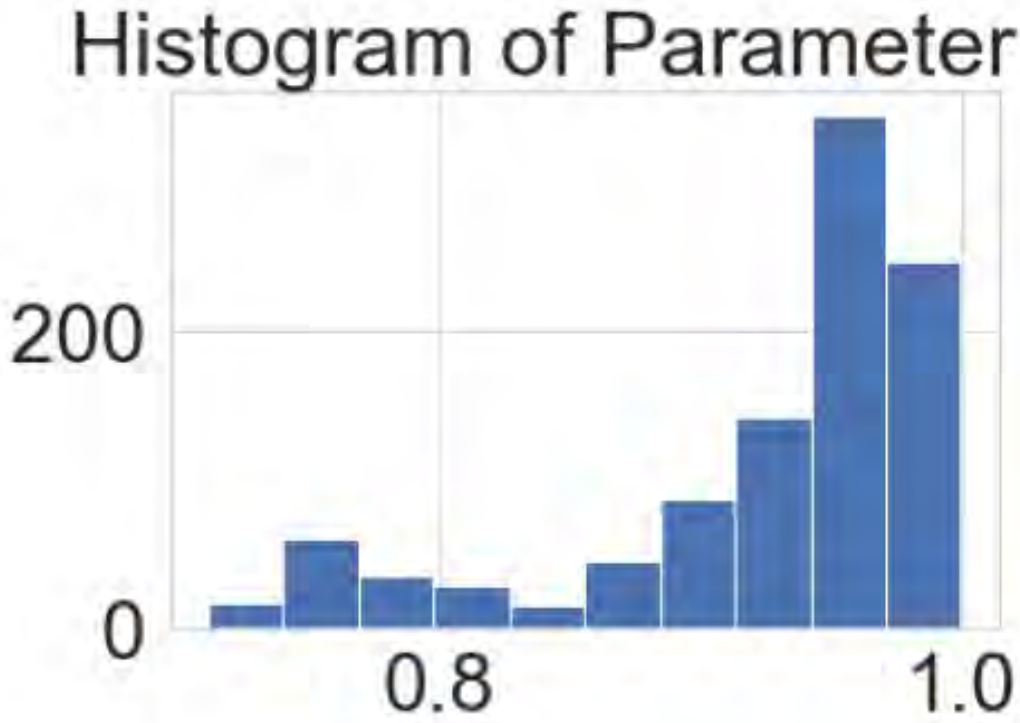
[7]:
```
r = accepted[0, after_burn_in:]
p = accepted[1, after_burn_in:]
r_average = np.average(r)
r_std = np.std(r)
p_average = np.average(p)
p_std = np.std(p)
```

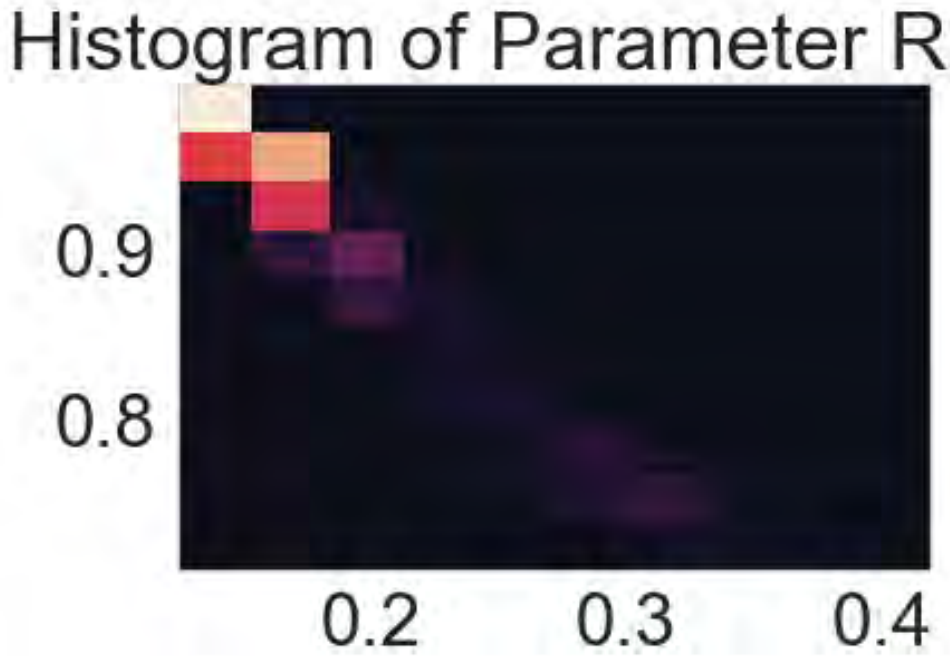
[8]:
```
plt.hist(r)
plt.title('Histogram of Parameter R')
plt.figure()
```

```
plt.hist(p)
plt.title('Histogram of Parameter P')
plt.figure()
plt.hist2d(r, p)
plt.title('Histogram of Parameter R & P')

print('R average: ', r_average)
print('R std: ', r_std)
print('P average: ', p_average)
print('P std: ', p_std)
```

```
R average:  0.1810073008676782
R std:  0.05068135528198024
P average:  0.9235651215065477
P std:  0.07044537382761534
```



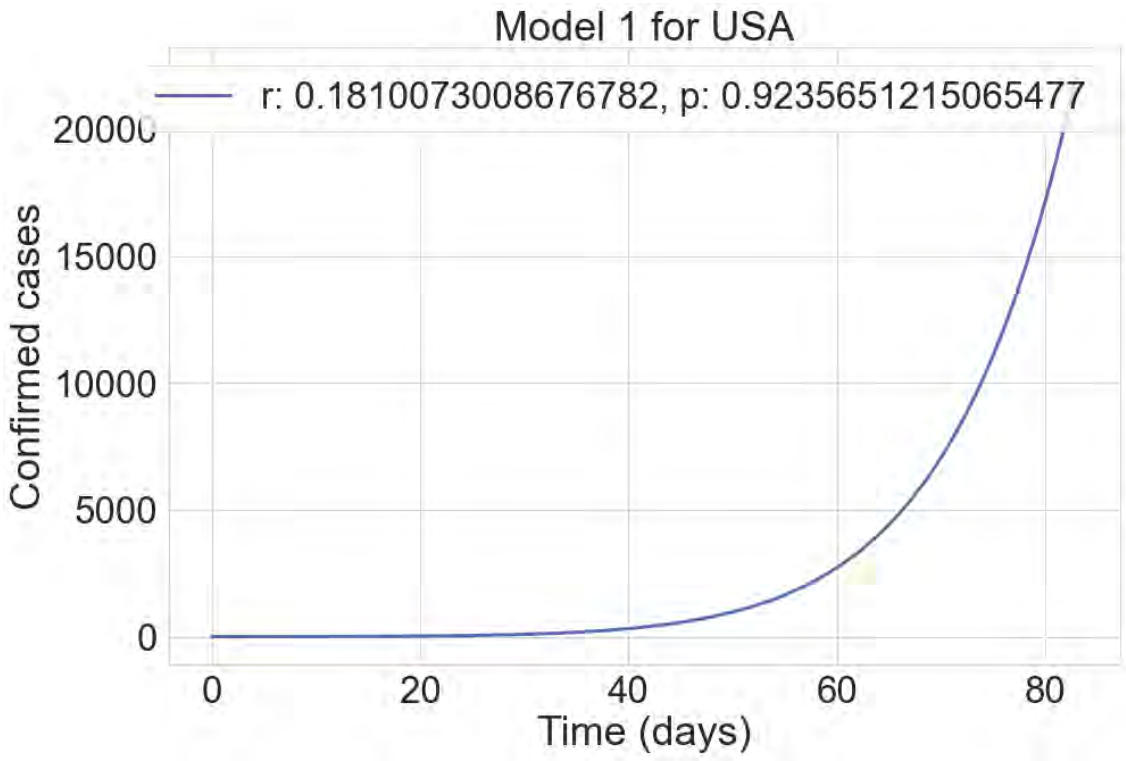

Histogram of Parameter P
200
0
0.8
1.0


Histogram of Parameter R & P
0.9
0.8
0.2
0.3
0.4

[9]:
```
t = [float(i) for i in range(84)]
T = 100.
C_0 = 1.
```

[10]:
```
C = model1([r_average, p_average], t, [C_0, T])
fig = plt.figure(figsize=(15., 10.))
plt.plot(t, C, label='r: {}, p: {}'.format(r_average, p_average))
plt.title('Model 1 for USA')
plt.xlabel('Time (days)')
plt.ylabel('Confirmed cases')
plt.legend()
```

[10]:
```
<matplotlib.legend.Legend at 0x104676cd0>
```



2.6 Validation

[11]:
```
validation_cases = validation_US[1].to_numpy()
validation_noise = np.std(calibration_cases)
cov = np.cov([r, p])
mu = np.array([r_average, p_average])
```

[12]:
```
transition_model1 = lambda theta: [np.random.normal(theta[0], r_std), np.random.
↪normal(theta[1], p_std)]

def prior_val(theta):
    if (theta[0] < 1.e-5 or theta[1] <= 1.e-5):
        return 1.e-8
    elif theta[1] >= (1 - 1.e-5):
        return 1.e-8
    else:
        return scipy.stats.multivariate_normal.pdf(theta, mean=mu, cov=cov)
```

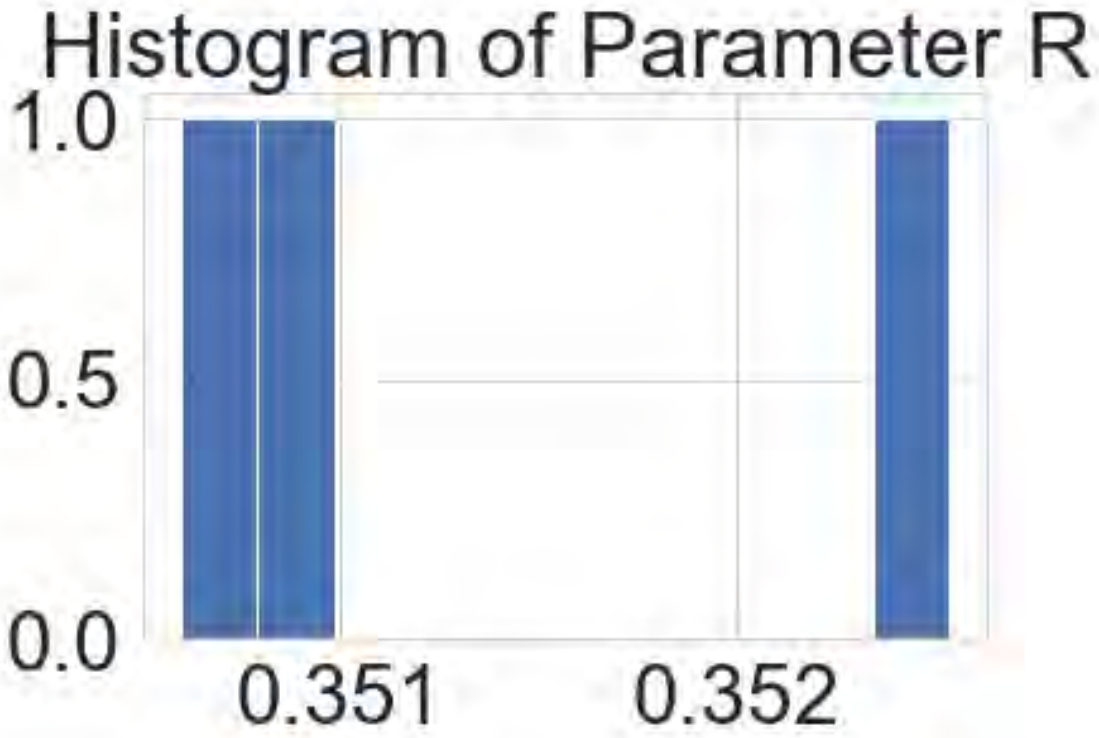
We run the Metropolis Hastings Algorithm 50,000 times, and we burn the first 80% of the accepted trials.

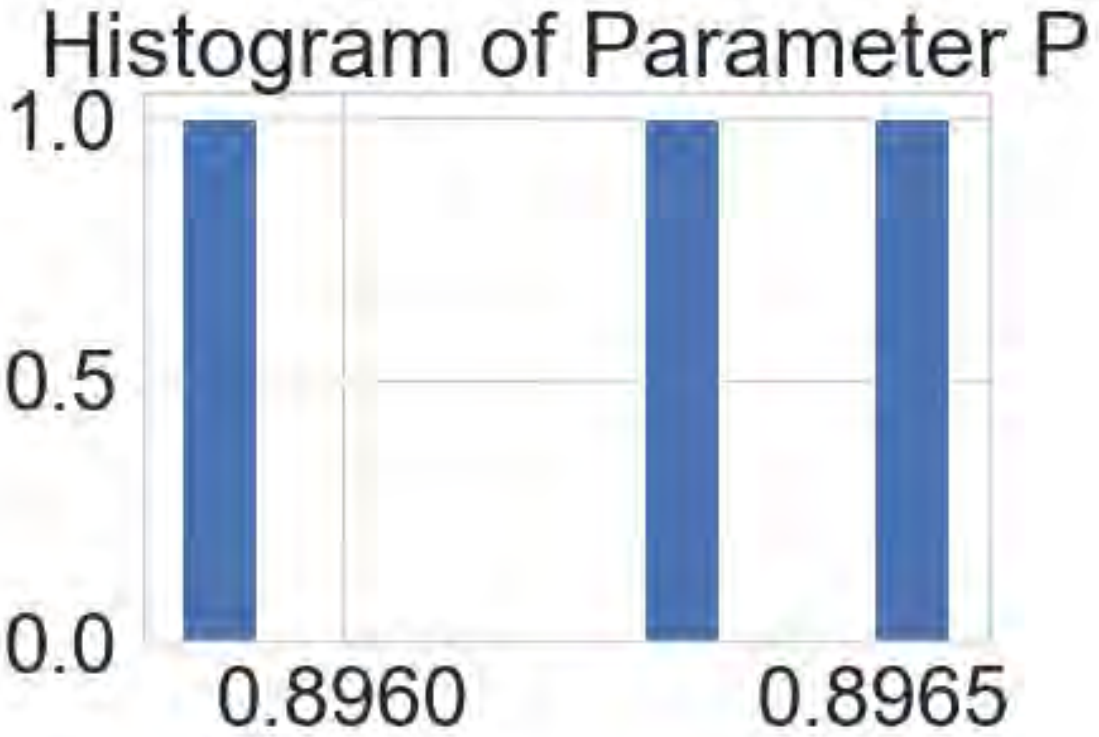
[13]:
```
val_accepted, val_rejected = metropolis_hastings(model1, log_lik, prior_val,␣
↪transition_model1, [r_average, p_average],
                                          n_iter, validation_times,␣
↪validation_cases, [1, 100], acceptance, validation_noise)
after_burn_in = int(len(val_accepted) * 0.8)
val_accepted = val_accepted.T
```

[14]:
```
val_r = val_accepted[0, after_burn_in:]
val_p = val_accepted[1, after_burn_in:]
val_r_average = np.average(val_r)
val_r_std = np.std(val_r)
val_p_average = np.average(val_p)
val_p_std = np.std(val_p)
```

[15]:
```
plt.figure()
plt.hist(val_r)
plt.title('Histogram of Parameter R')
plt.figure()
plt.hist(val_p)
plt.title('Histogram of Parameter P')

print('R average: ', val_r_average)
print('R std: ', val_r_std)
print('P average: ', val_p_average)
print('P std: ', val_p_std)
```

```
R average:  0.3513542586919303
R std:  0.000850071655461424
P average:  0.896248239690765
P std:  0.0002957058087295114
```


Histogram of Parameter R
1.0
0.5
0.0
0.351
0.352


Histogram of Parameter P
1.0
0.5
0.0
0.8960
0.8965

[20]:
```
t = [float(i) for i in range(84)]
T = 100.
C_0 = 1.
```

[26]:
```
C = model1([val_r_average, val_p_average], t, [C_0, T])
print('r2_score: ', r2_score(C[50:], validation_cases))
```

```
r2_score:  0.9725454490704682
```

2.7 Prediction

We take the validation parameters and their standard deviations, then we sample a prediction.

[16]:
```
def model1_prediction_sampler(theta, theta_std, iterations):
    r = theta[0]
    p = theta[1]

    r_std = theta_std[0]
    p_std = theta_std[1]

    T = 100
    C_0 = 1

    predictions = []
    for i in range(iterations):
        r_i = np.random.normal(r, r_std)
        p_i = np.random.normal(p, p_std)
        prediction = model1([r_i, p_i], [T], [C_0])
        if prediction in [np.inf] or prediction < 0:
            continue
        elif prediction > 1e8:
            continue
        predictions.append(prediction)

    return predictions
```

We run the prediction 50,000 times, and we burn the first 80% of the accepted trials.

[17]:
```
predictions = model1_prediction_sampler([val_r_average, val_p_average],␣
↪[val_r_std, val_p_std], n_iter)
```

[18]:
```
predictions = np.array(predictions)
predictions = predictions.T
after_burn_in = int(len(predictions) * 0.8)
predictions = predictions[after_burn_in:]
```

[19]:
```
print(int(model1([val_r_average, val_p_average], [T], [C_0, T])))
print(int(np.average(predictions)))
```

```
2685315
2686550
```

The prediction from the model using just the validation parameters is 2,685,315 total cases on the 100th day in the US. The prediction from the model using the samples is 2,668,550 total cases on the 100th day in the US.

Chapter 10

COVID-19 Trend Analysis for the US

Sheroze Sheriffdeen
CSE397 - Foundations of Predictive Computational Science
University of Texas at Austin

April 27, 2020

1 Problem

The goal of this assignment is to fit the confirmed COVID-19 cases as a function of date using the generalized growth model given by,

$$\frac{dC(t)}{dt} = rC(t)^p$$

where, $t \in [0, T]$ is the time (in units of days), $r \geq 0$ is the growth rate, $p \in [0, 1]$ is the 'deceleration of growth' parameter. [1] For $0 < p < 1$, the above equation can be solved:

$$C(t) = \left(\frac{r}{m}t + C_0^{1/m}\right)^m$$

where $m = 1/(1-p)$ and $C_0 = C(0)$ is the initial condition.

2 Model Description and Data

The dataset for confirmed COVID-19 in USA is used for this analysis. The dataset consists of timestamps $\bar{t} = (t_1 = 0, t_2 = 1, ..., t_N = 83)$, where $N = 84$, and corresponding total confirmed cases $Y(\bar{t}) = (Y_1, Y_2, ..., Y_N)$. The model prediction is $C(\bar{t}) = (C(t_1), C(t_2), ..., C(t_N))$ where $C(t_i)$ is evaluated with learned parameters $\theta = \{r, p\}$.

2.1 Model Particulars

2.1.1 Parameter Priors

A uniform prior is used for both parameters with $r \sim \mathcal{U}[0.0001, 0.99]$ and $p \sim \mathcal{U}[0.01, 10]$.

2.1.2 Model Likelihood

The observation model assumes that the confirmed cases are corrupted with Gaussian noise, i.e. $Y(\bar{t}) = Y^{\text{true}}(\bar{t}) + \epsilon$ where $\epsilon(t) \sim \mathcal{N}(0, \sigma(t)^2)$. Thus, the likelihood is evaluated as:

$$\mathbb{P}(C(\bar{t})|Y(\bar{t}), \Sigma) = \frac{1}{2\pi^{T/2}|\Sigma|^{1/2}} \exp\{-\frac{1}{2}(C(\bar{t}) - Y(\bar{t}))^T \Sigma^{-1}(C(\bar{t}) - Y(\bar{t}))\}$$

where $\Sigma = \text{diag}(\sigma(0)^2, \sigma(1)^2, \ldots, \sigma(T)^2)$. The noise variance $\sigma^2(t)$ was chosen to be 1 + 20t.

[1] Viboud et al 2015 Viboud, C., Simonsen, L. and Chowell, G., 2016. A generalized-growth model to characterize the early ascending phase of infectious disease outbreaks. Epidemics, 15, pp.27-37.

2.2 Model Calibration

The first $N_c = 60$ days are used to calibrate the above model. Metropolis-Hastings algorithm is used to perform Markov Chain Monte Carlo sampling of the posterior. The posterior mean of the parameters are used to validate the model and predict cases at $t = 100$. The posterior variance is used to compute uncertainty of the prediction. PyMC3 implementation of the Metropolis-Hastings algorithm was used to generate the results for this assignment. Two parallel chains with 20000 samples each was used to form the posterior. 3000 tuning samples were used to tune the hyperparameters of the model such as the initial values of the chains and proposal variance.

3 Results

The predicted parameters using the Metropolis-Hastings algorithm is described below:

Parameter	Posterior Mean	Posterior Std. Dev.	Effective Samples
Growth rate (r)	0.175	0.0012	553
Deceleration of growth (p)	0.989	0.0012	458

The mean squared validation error of the model using the posterior mean as the parameters is 0.5 million.

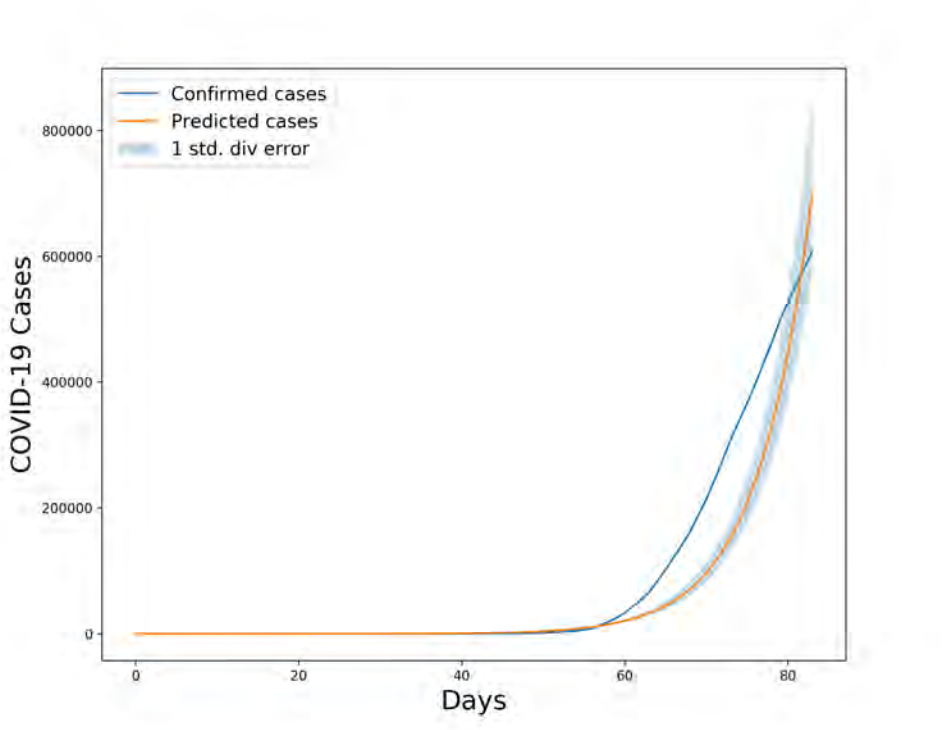


Figure 1: Predicted cases calibrated with 60 days of data with uncertainty visualized

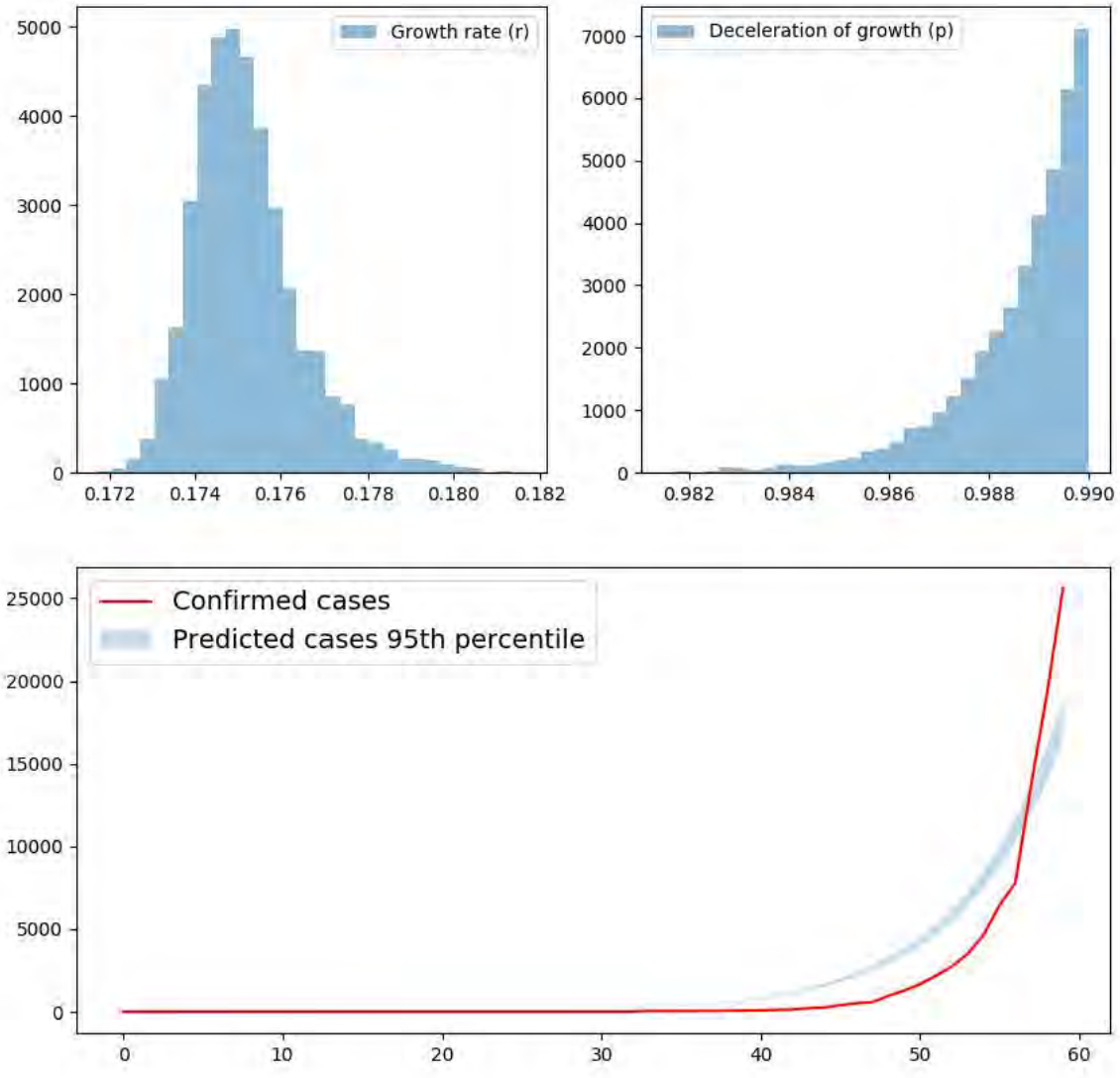


Figure 2: Posterior sample histogram of parameters

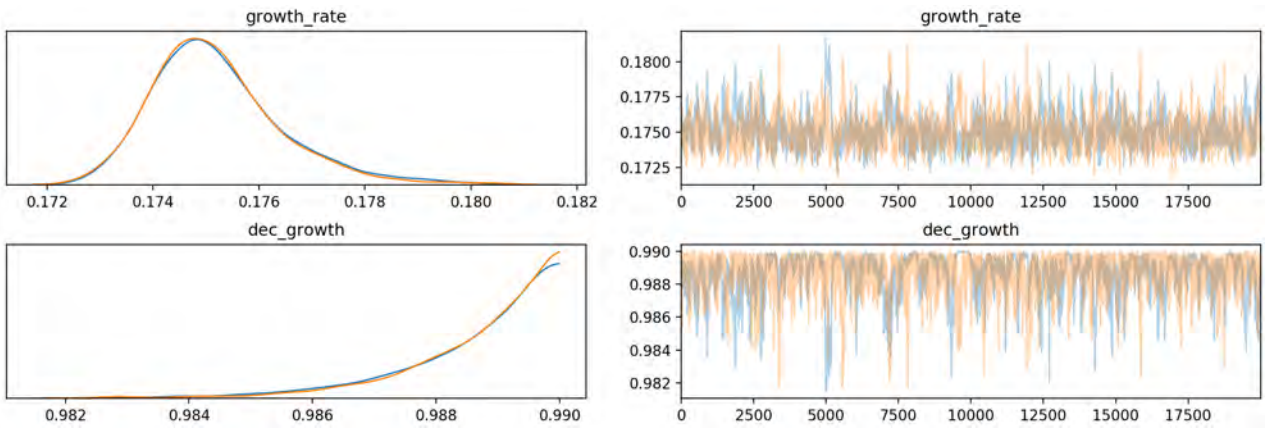


Figure 3: Trace plot of parameter samples shows a well mixed chain. The proposal density is scaled during the 3000 tunings steps.

Finally, the calibrated and validated model predicts 7.5 million cases at $t = 100$ days. The uncertainty of this prediction is computed by solving the model for 1 standard deviation below and above for the parameters, leading to an optimistic prediction of 5.8 million cases and a pessimistic prediction of 9.7 million cases.

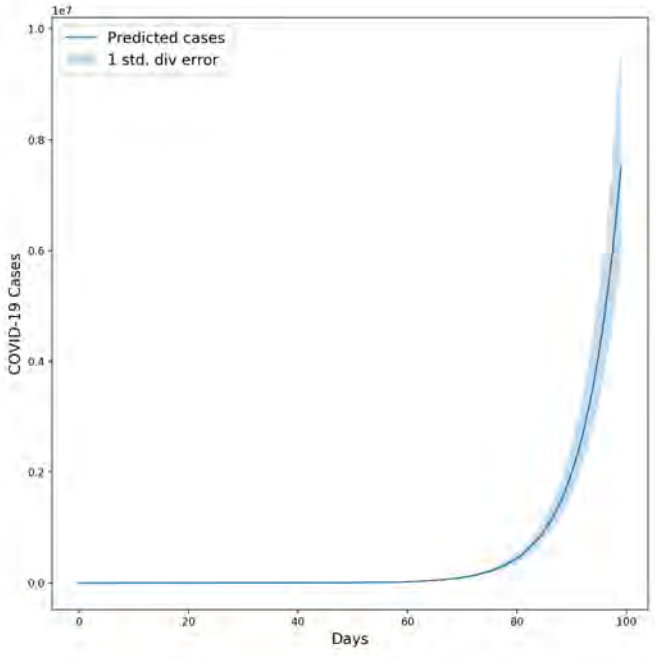


Figure 4: Model prediction of future cases: 7,503,814 cases at 100 days with 1 standard deviation bounds being [5,837,875, 9,718,257]

Chapter 11

Covid Simulated Annealing

Chase Tessmer

May 2020

1 Model

I decided to use a logistic curve model because it is derived by integrating a VERY simple population growth differential equation which seems appropriate for virus spread when we only have information on virus growth. Also, the parameter L will be the final virus count after the pandemic is over if the model is valid (Figure 1).

$$p = \frac{L}{1 + e^{-r(x-p)}} \tag{1}$$

L final case count
r logistic growth rate
p inflection point

2 Using the Code

The model can be changed by changing the forward3 function.
All of the parameters are under Parameters in the code.
You can change the starting value of the parameters and the standard deviation of the distributions the MCMC algorithm will be sampling from along with the number of iterations and the acceptance delay which determines how long the model is allowed to settle into the Metropolitan-Chain distribution.
Finally you can edit the data covariance matrix to change the "emphasis" on different data points. I edited it to have more emphasis on the final data point and less emphasis on the beginning.
You can also change the start and stop temperatures if you want to use simulated annealing. Just keep in mind that the temperatures are in log base.
You can also change the calibration and validation split. I went for the first sixty datapoints in calibration and the final twenty-three in validation.
The parameters are selected by averaging the accepted parameters after the acceptance delay. The standard deviation is selected by taking the standard deviation of the accepted parameters after the acceptance delay.

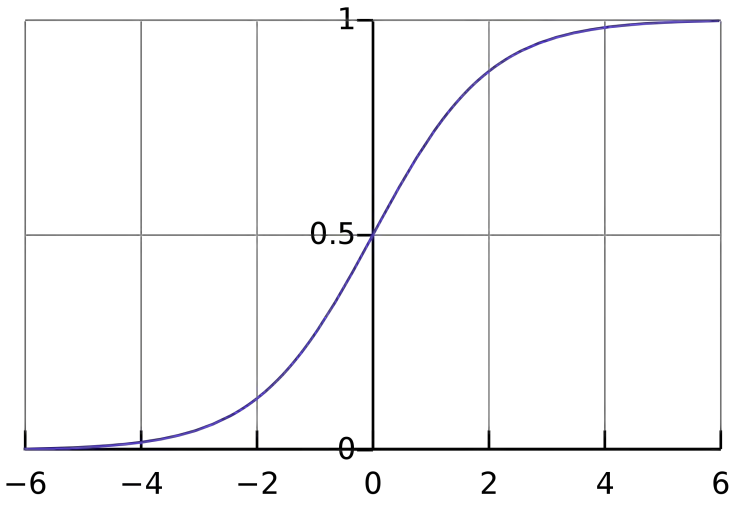


Figure 1: By Qef (talk) - Created from scratch with gnuplot, Public Domain, https://commons.wikimedia.org/w/index.php?curid=4310325

3 Calibrated Model

I calibrate on the first 60 datapoints for ten different models and whichever model performs the best on the entire dataset, I take that one. Generally you want to have about a 20% acceptance rate to say your model has converged. I generally have about a 10% acceptance rate, but I think that's good enough. Finally the quantity of interest is the number of cases at 100 days, and its mean and standard deviation are found in the exact same way as the parameters. I calculated the QOI for every set of accepted parameters and also took the mean and standard deviation of that.

According to the model, the final case count will be 10,930 cases ±56 at 95% confidence . I only fit the Korean data because the US and Japan cases hadn't hit an inflection point and trying to predict them simply from the data doesn't seem realistic. One big problem I had is that most of the calibrated models predicted an $L < 1$ which would mean that the final case count would be less than the current case count, which is obviously unreasonable. Fortunately, the validation step removed all of these models.

QOI (100th day cases) | 10,903 ± 56 @ 95%

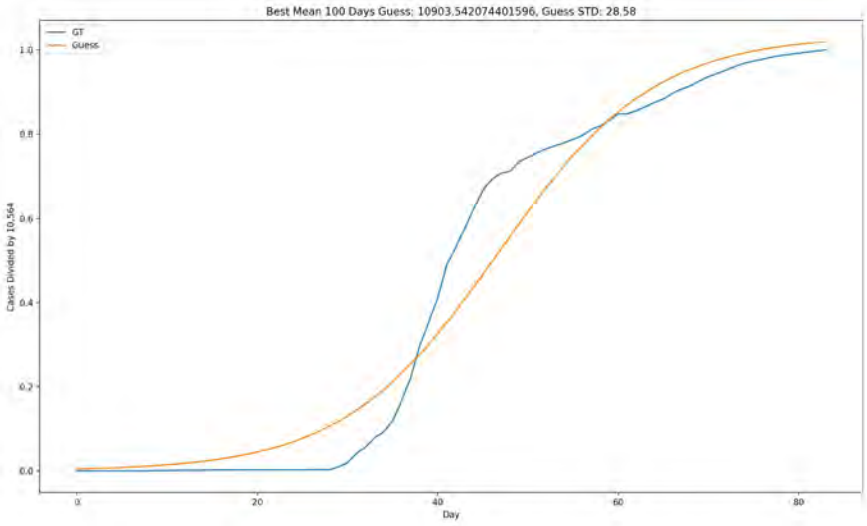


Figure 2: Best Model. L=1.035

Chapter 12

BAYESIAN PREDICTION PRACTICE FOR COVID-19

Yang, Christine

Course: EM397 Foundation of Predictive Computational Science
Instructor: Dr. J. Tinsley Oden

The University of Texas at Austin
Aerospace Engineering
and Engineering Mechanics
Cockrell School of Engineering

Table of Contents

Objective:

The objective of this write-up is to utilize Bayesian approach to predict the confirmed cases of COVID-19 until 100 days in the United States and South Korea. For the example of United States, the procedure is described in detailed along with the results; for the example of South Korea, since the same procedure and approach are followed, the write-up focus on presenting the results. The python coding methodology is strongly based on Moukarzei's journal paper on the toward-data-science website [1].

Methodology:

Compared to the frequentists, which only considers a limited case of repeated measurements, Bayesian approach represents the belief about the probabilities of the future (posterior) based on the previous events (prior beliefs) and current events (evidence). Bayesian approach is powerful in many areas of science and engineering. For example, it can be used to examine the structural health monitoring (SHM), finding the most probable structural model based on incomplete noisy modal data (e.g. partial mode shapes of the some of the modes of a structural system) [2].

Bayesian theorem can be expressed as equation 1.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B0} \quad \textbf{(1)}$$

where,

$P(A|B)$ represents the probability of A occurring given that event B has occurred.

P(B|A) is the likelihood

P(A) is the prior

P(B) is the evidence

Since the goal is to sample the posterior distribution, the process can become very tedious due to the complexity of the model involving multidimensional integration. Thus, MCMC (Markov Chain Monte Carlo) can be implemented to draw sample from conditional posterior distributions. In addition, Metropolis-Hastings, an algorithm suitable for high-dimensional spaces, is utilized to implement MCMC.

The training data from the observable data will be used for calibration first. The initial proposal prior and the initial guess values will be implemented into the MCMC algorithm. After the MCMC is implemented, a certain amount of the beginning of iterations will be "thrown away", which is called "burn-in". The average of the accepted mean values after burn-in will be used as the new initial guess values, and the new prior will be defined as the multivariate Gaussian density function. The testing data will be implemented into the new defined MCMC to validate the model. Again, the validated parameters used to determine QoI (Quality of Interest) will be the mean values after burn-in.

Process:

1. Import Data

The data of the confirmed cases of the country includes 84 days. The data of the first 50 days is used as training data for calibration, and the rest of 34 days is used for validation, as Figure 1.

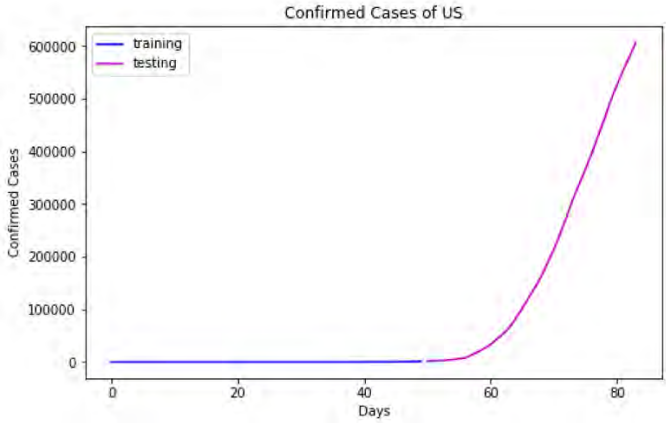


Figure 1. Confirmed Cases of the US from the Data.

2. Define the Model

The model is given as:

$$C(t) = aexp\left[b\left(1 - \frac{1}{1-\left(1-\frac{t}{T}\right)^{p}}\right)\right] + C_0 \quad \textbf{(2)}$$

where,

C is the confirmed cases at the certain time
C_0 is the initial condition, (i.e. the cases at the first day)
a and b are model parameters
T is the final day at which we want to model prediction
p is assumed to be equal to 4

3. Define the PDF (Probability Distribution Function) and the Transition Model

Since the Metropolis-Hastings algorithm requires a proposal distribution $Q\left(\frac{\theta_{new}}{\theta_{current}}\right)$, which is also called transition model, to draw samples from posterior distribution.

The Gaussian distribution is used:

$$Q\left(\frac{\theta_{new}}{\theta_{current}}\right) = \mathcal{N}(\mu = \theta_{current}, \sigma) \quad \textbf{(3)}$$

It should be noted that σ only affects the computational time of the algorithm, and it does not affect the results.

4. Define the Prior for Calibration

The goal is to find the proper a and b for the model. The prior θ are enforced as nonnegativity as equation 3.

$$\pi_{prior}(\theta) = \mathcal{X}_A(\theta) \quad \textbf{(4)}$$

where, θ = [a, b], and A = $[0, \infty] \times [0, \infty]$.

```
def prior(theta):
    if (theta[0]<=0 or theta[1]<=0):
        return 0
    else:
        return 1
```

5. Define the Likelihood Function

Next step is to define the likelihood:

$$\pi_{likelihood}(Y|\theta) \propto \boldsymbol{f} = \exp\left[-\frac{||Y-C(\theta)||^2}{2\sigma^2}\right] \quad \textbf{(5)}$$

where, Y is the observable data, σ is the standard deviation of the noise of the data, and $C(\theta)$ is the model output.
It should be noted that the logarithm of the likelihood will be used.

```
def log_like(theta,data):
    t = observation[:,0]
    Y = observation[:,1]
    C = model(theta,t,model_param)
    return (-1. / (2. *noise_sigma * noise_sigma)) * np.dot(Y - C, Y-C)
```

6. Define the Acceptance Rule

Using Baye's theorem, the following ratio is computed,

$$ratio = \frac{P(D|\theta')P(\theta')}{P(D|\theta)P(\theta)} \quad (6)$$

The random number r will be drawn from a uniform distribution in [0,1]. If the ratio is larger than this random number, then the new algorithm will accept the new parameters.

```
#Defines whether to accept or reject the new sample
def acceptance_rule(x,x_new):
    if x_new > x:
        return True
    else:
        accept = np.random.uniform(0,1)
#since we use log likelihood, we need to exponentiate in order to compare to the random number
        return (accept < (np.exp(x_new-x)))
```

7. Employ the Metropolis-Hastings Algorithm

Given the likelihood function, the proposal distribution, the initial guess of the parameters, θ, the for loop will use the acceptance rule to determine if the new parameters can be accepted or rejected.

```
def metropolis_hastings(likelihood_computer, prior, transition_model,param_init, iterations, data, acceptace_rule):
#model: reference to the output model
#likelihood  model(x,data): returns the likelihood that these parameters generated the data
#transition_model(x): a function that draws a sample from a symmetric distribution and returns it
#param_init: a starting sample
#iterations: number of accepted to generated
#data: the data we wish to model
x = param_init
accepted = []
rejected = []

for i in range(iterations):
   x_new = transition_model(x)
   x_lik = likelihood_computer(x,data)
   x_new_lik = likelihood_computer(x_new,data)
   if (acceptance_rule(x_lik + np.log(prior(x)),x_new_lik + np.log(prior(x_new)))):
      x = x_new
      accepted.append(x_new)
   else:
      rejected.append(x_new)
return np.array(accepted),np.array(rejected)
```

Case: US

Calibration

a. The initial guess of the parameters $\theta = [a, b] = [607670, 100]$. The number 607670 is the maximum value in the observable data. In addition, the standard deviation in the likelihood in equation 5 is set to be 100.

```
init_guess = [np.max(observation),100]
noise_sigma = 100.
```

b. The proposal distribution is defined as equation (3), and the $\sigma = [100,1]$.

```
transition_model_calibration = lambda x:np.random.normal(x,[100,1] ,(2,))
```

c. Results:

Figure 2 shows both accepted and rejected samples using the training data for total 100,000 iterations.

```
accepted_calibration, rejected_calibration = metropolis_hastings(log_like, prior, transition_model_calibration ,
init_guess, iteration = 100,000, data_calibration, acceptance_rule)

plot_sampling(accepted_calibration, rejected_calibration)
```

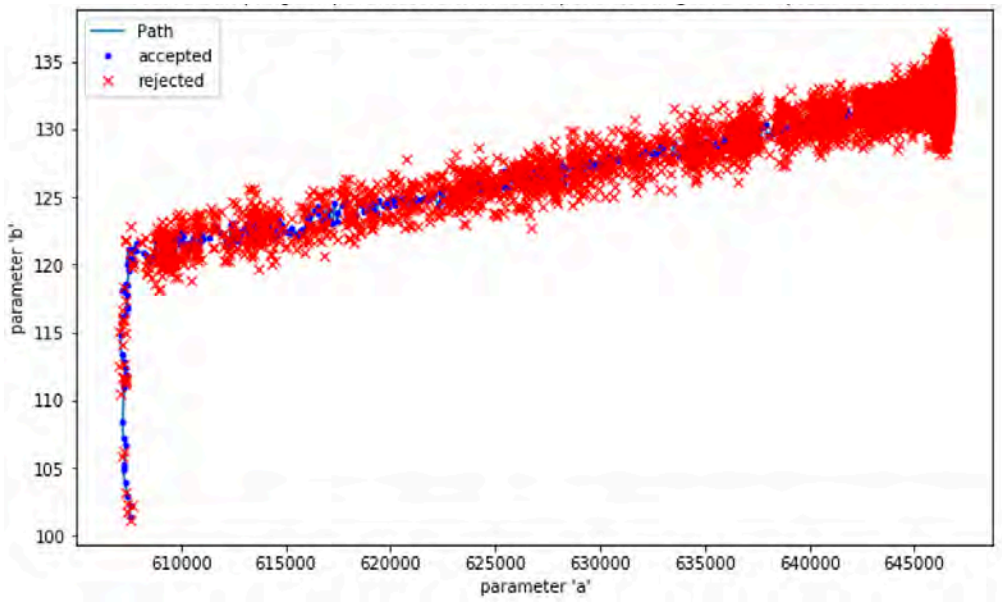


Figure 2. MCMC Sampling for Parameters of Training Data. All Samples Are Shown.

It was also found that 1841 iterations are accepted. This value changes every time the algorithm is performed.

```
print(accepted_calibration.shape)
(1841,2)
```

The burn-in percentage is set to 50%, and the final mean value of the calibrated parameter is:

$$\theta = [a, b] = [6.46295377\mathrm{e}+05\ 1.32483842\mathrm{e}+02]$$

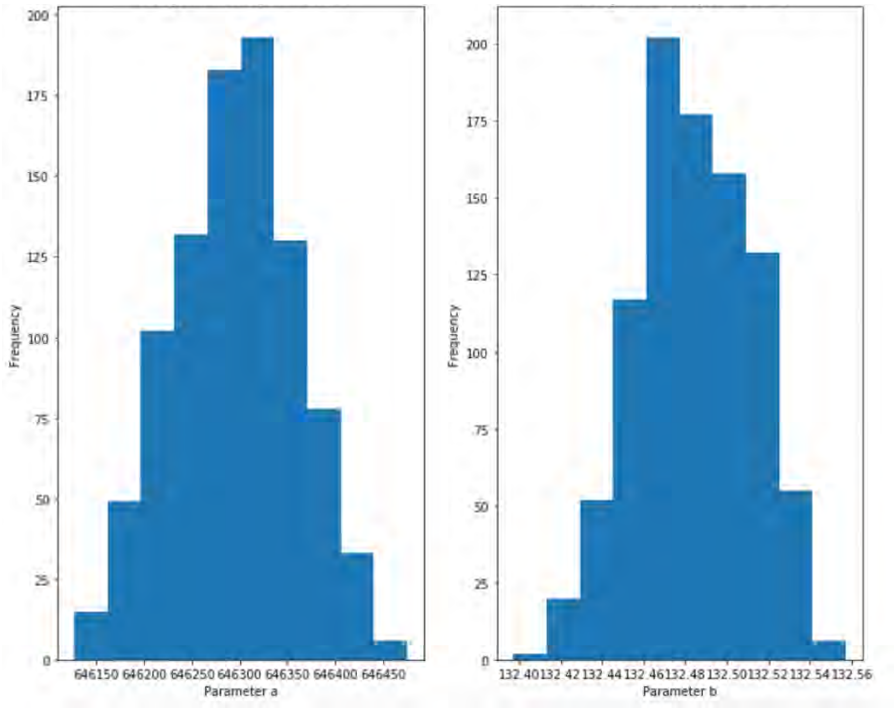


Figure 3. Histogram of the Parameters after Calibration and Burn-In.

Figure 3 shows the histogram of the calibrated parameters after burn-in. They both look like following the Gaussian distribution, and it can be concluded that they converge well.

Validation

a. The new initial guess now is set to be the mean values of calibrated data after burn-in.

```
prior_new = mean_calibration
```

b. From equation 3, again, the new prior should be defined as multivariate normal distribution.

```
def posterior_new_prior(theta):
    return multivariate_normal.pdf(theta, mean = mean_calibration, cov = covariance_calibration)

transition_model_validation = lambda theta:multivariate_normal.rvs(theta, covariance_calibration )
#generate random variation in multivariate normal distribution
```

c. Again, MCMC is implemented with the testing data,

```
accepted_validation, rejected_validation = metropolis_hastings(log_like, posterior_new_prior,
transition_model_validation , prior_new, iteration = 100000, data_validation, acceptance_rule)
```

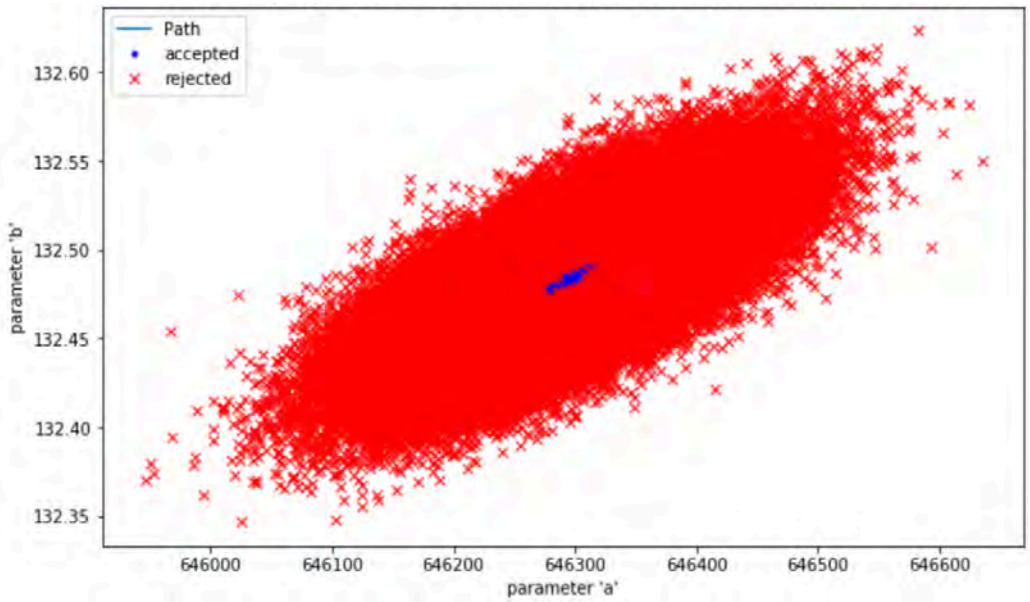


Figure 4. MCMC Sampling for Parameters of Testing Data. All Samples Are Shown.

Figure 4 shows that most accepted parameters are around [646300, 132.45]. For the total of 100,000 iterations, the total of 39747 iterations are accepted.

```
print(accepted_validation.shape)
(39747,2)
```

The burn-in percentage is set to be 25%. And the mean values of the parameters are:

$$\theta = [a, b] = [6.46294957\text{e}+05\ 1.32483397\text{e}+02]\ ,$$

Which is approximately equal to the values obtained from the calibration.

Figure 5 presents the histogram of the parameters. They follow the Gaussian distribution closer to the calibration data.

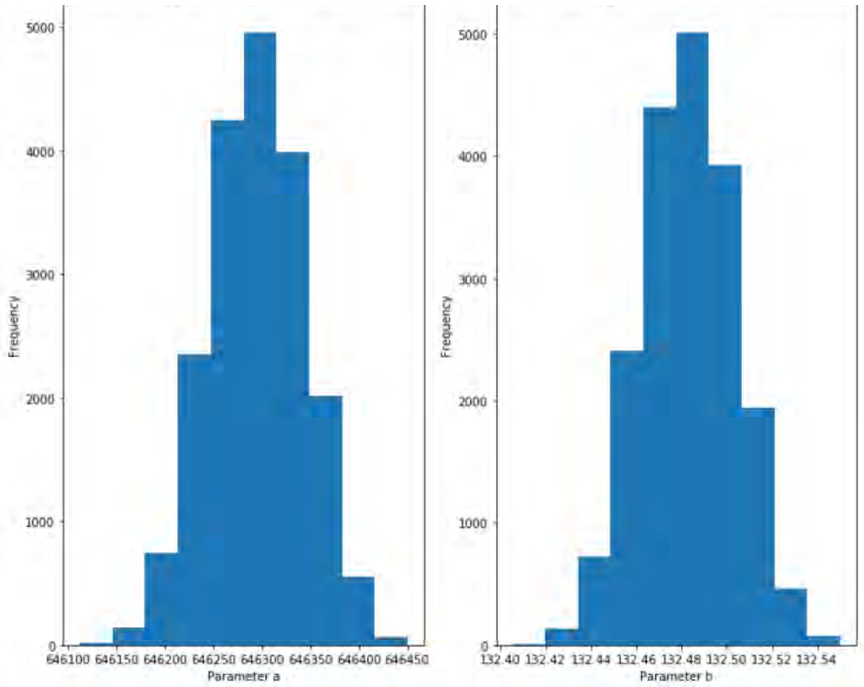


Figure 5. Histogram of the Parameters after Validation and Burn-In.

Conclusion: Fitting Curves and QoI

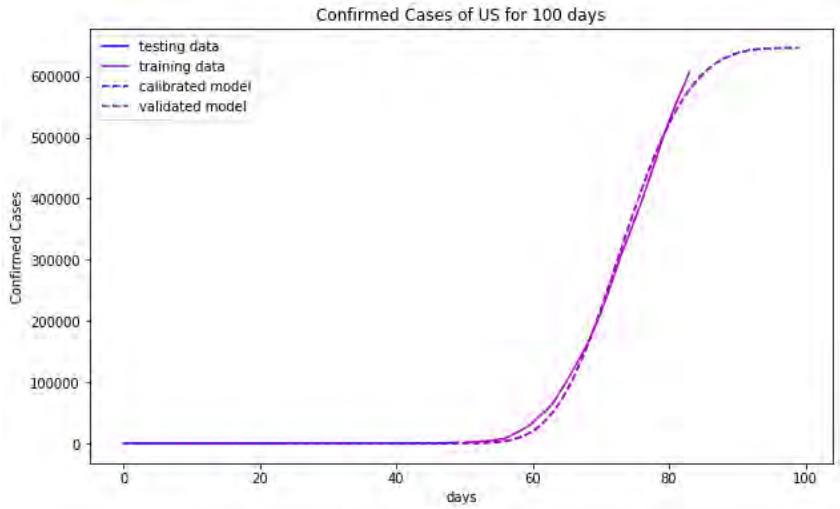
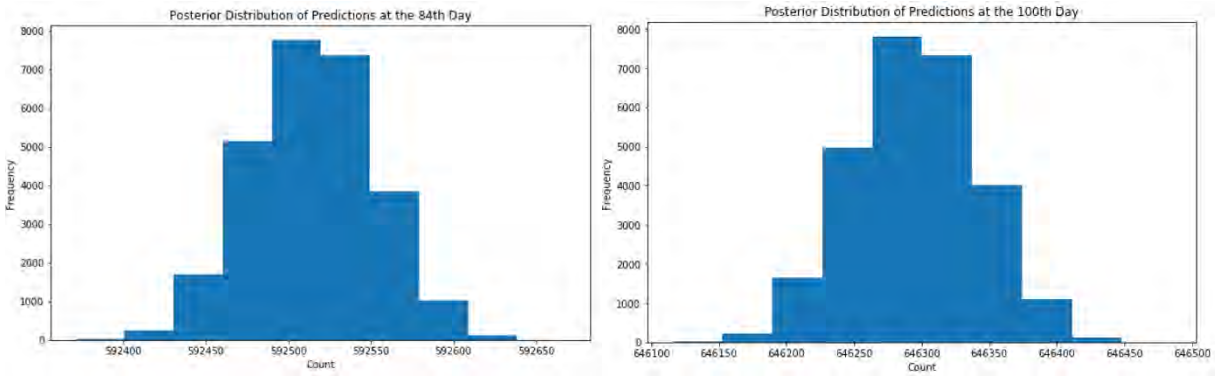


Figure 6. Curve Fitting from Training and Testing data Compared to the Observable Data

Figure 6 shows the curve fitting of the training and testing data. As seen from the previous sections, the mean values of the parameters are almost the equal from training and testing data, thus, the curve-fitting after the 84th day are the same. The validated data after 50 days matches the observable data well.

Figure 7. shows the posterior distribution of the predictive model at the 84th day. From the observable data, the true value is 607670 cases, which is around 2.5% difference from the predictive model. Figure 8 represents the probability distribution of the model at the 100th day.



a. **b.**

Figure 7. Posterior Distribution of Predictions at the **(a)** 84th Day **(b)** 100th Day

Case: South Korea

The model is the same as before, and the same procedure is used. Part A is to estimate the confirmed cases until the 100 days, and part B is for 200 days.

A. 100 Days

Calibration

```
init_guess = [np.max(observation),10]
noise_sigma = 1.
transition_model_calibration = lambda x:np.random.normal(x,[1,0.001] ,(2,))
accepted_calibration, rejected_calibration = metropolis_hastings(log_like, prior, transition_model_calibration , init_guess,
iteration = 100000, data_calibration, acceptance_rule)
print(accepted_calibration.shape)
(14810,2)
```

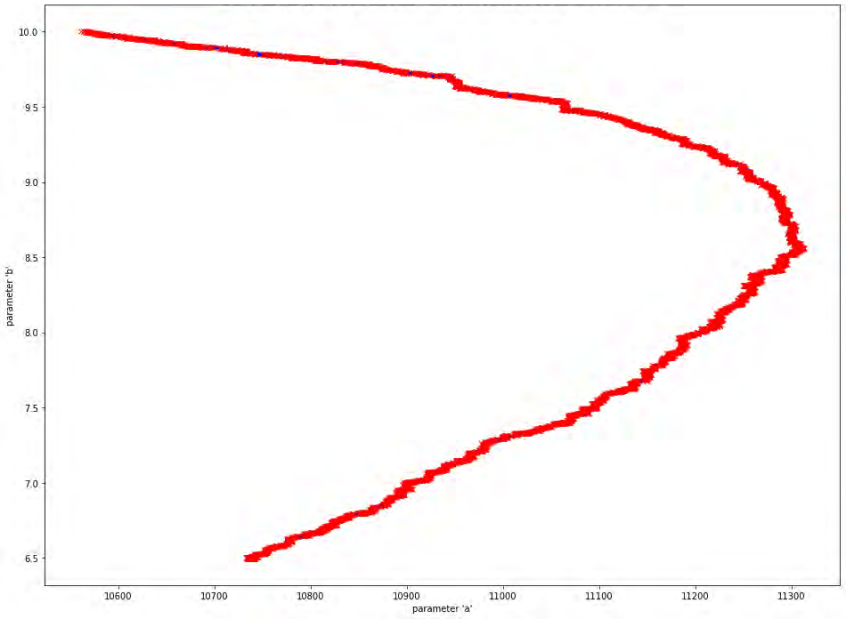
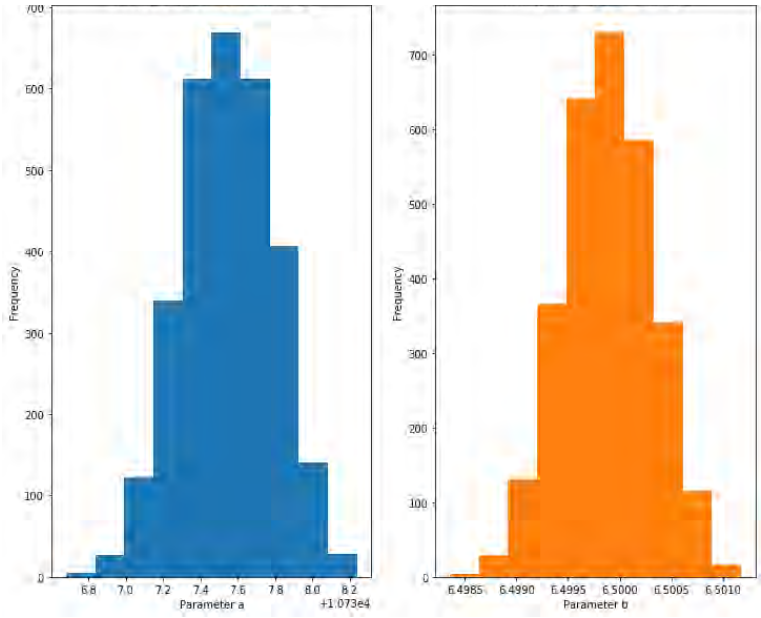


Figure 8. MCMC Sampling for Parameters of Training Data. All Samples Are Shown.



a. **b.**

Figure 9. Histogram of the Parameters after Calibration and Burn-In. (a) parameter "a" (b) parameter "b"

Burn-in = 0.5
print(mean_calibration)
[1.07375478e+04 6.49988446e+00]

In the calibration, less than 20% of the iterations are accepted. The histogram follows the trend of the Gaussian distribution. From the histogram and the mean values, the parameters converge to 10737.5 and 6.5.

Validation

Figure 10 shows all samples from the validated data after implementing the mean values of parameters from the calibration as the new initial guess in the MCMC. Again, the proposal distribution is assumed to be multivariate normal distribution.
Figure 11 shows the histogram of the samples from the testing data.

Burn-in = 0.3
print(mean_validation)
[1.07375447e+04 6.49988355e+00]
print(accepted_calibration.shape)
(39664,2)

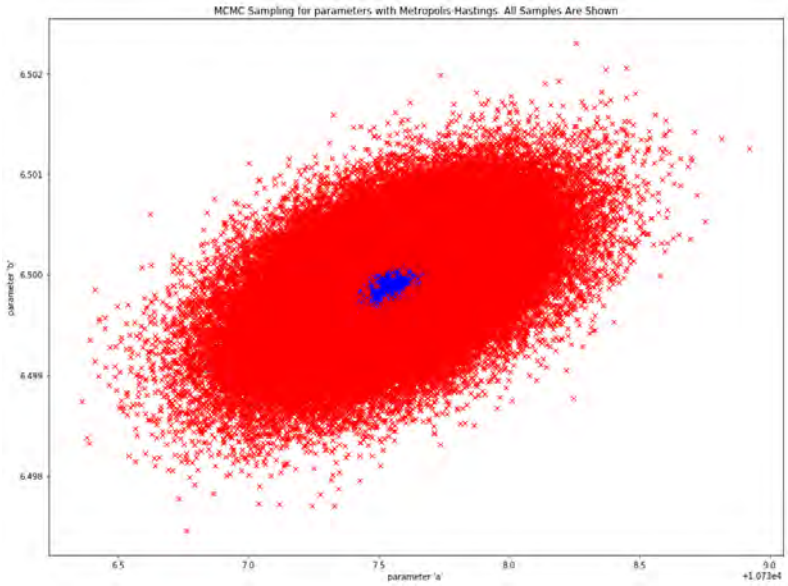


Figure 10. MCMC Sampling for Parameters of Testing Data. All Samples Are Shown.

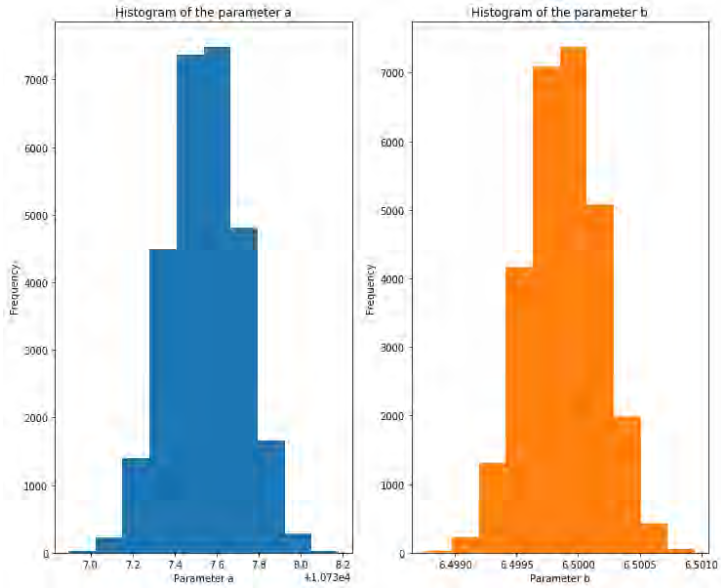


Figure 11. Histogram of the Parameters after Validation and Burn-In.

Conclusion: Fitting Curves and QoI

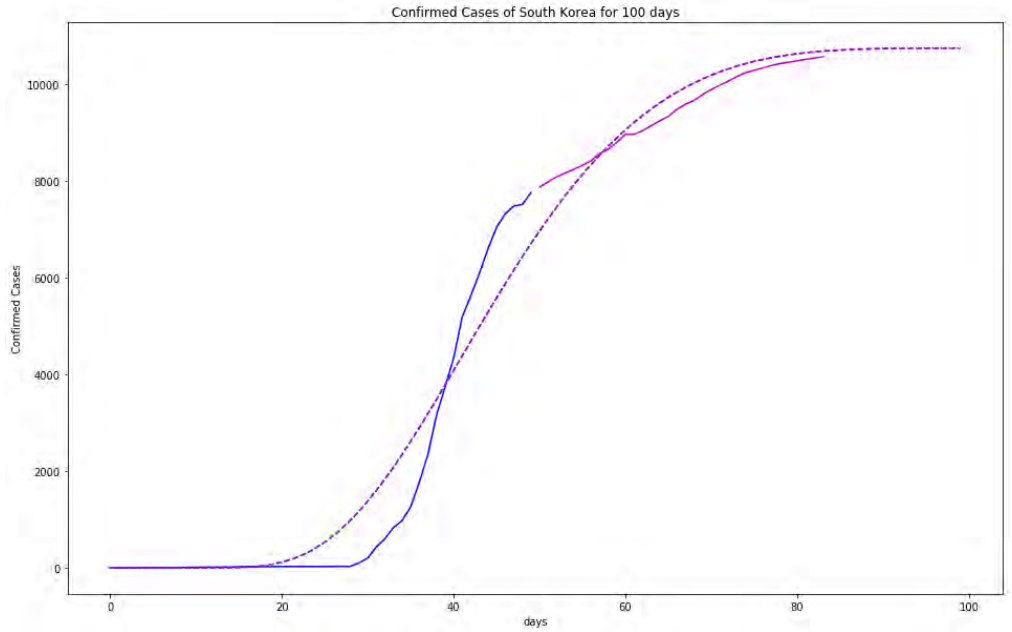
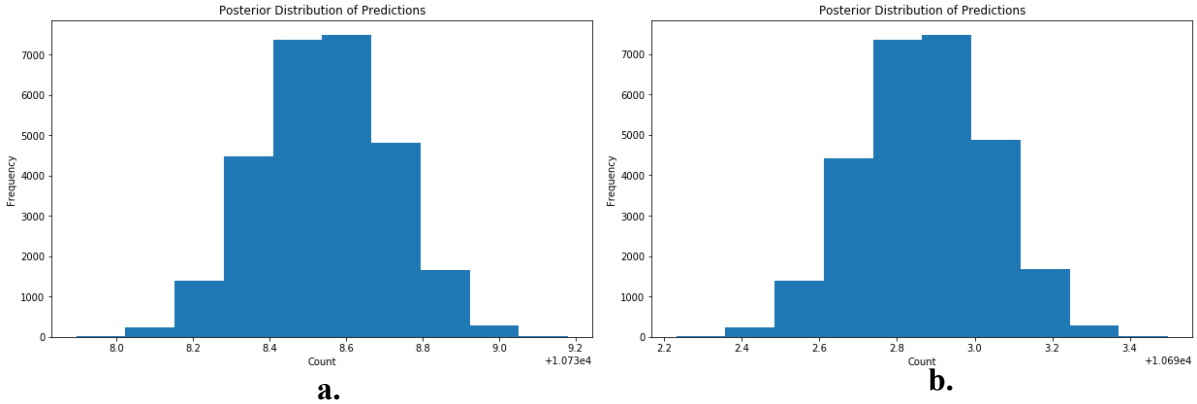


Figure 12. Curve Fitting from Training and Testing data Compared to the Observable Data



a. **b.**

Figure 13. Posterior Distribution of Predictions at the (a) 100th Day (b) 84th Day

At the 84th day, the error from the true value is around 1.18% which could be considered acceptable. From the figure 12, there seems to be larger discrepancy in the training data, but, after 60 days, the curve-fitting correlate with the testing data better.

B. 200 Days Calibration:

```
init_guess = [15000,2]
noise_sigma = 1.
transition_model_calibration = lambda x:np.random.normal(x,[1,0.001] ,(2,))
accepted_calibration, rejected_calibration = metropolis_hastings(log_like, prior, transition_model_calibration , init_guess,
iteration = 100000, data_calibration, acceptance_rule)
print(accepted_calibration.shape)
(3821,2)
```

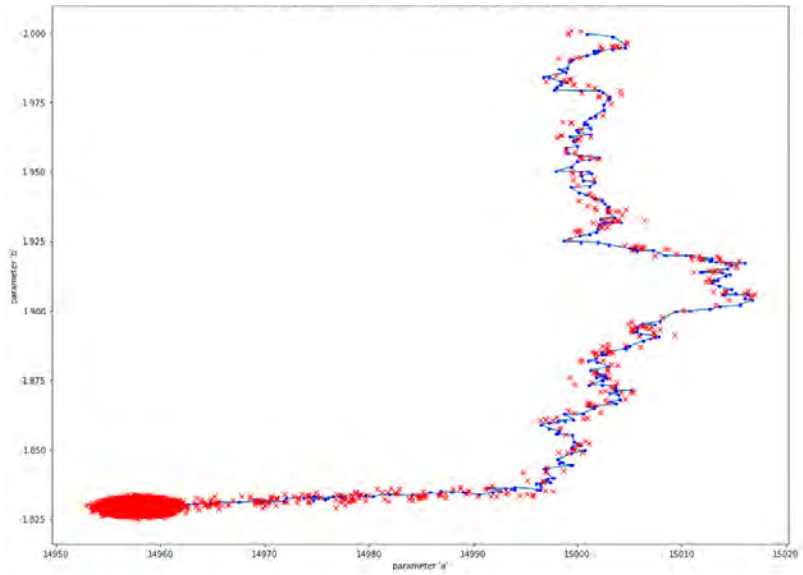


Figure 14. MCMC Sampling for Parameters of Training Data. All Samples Are Shown.

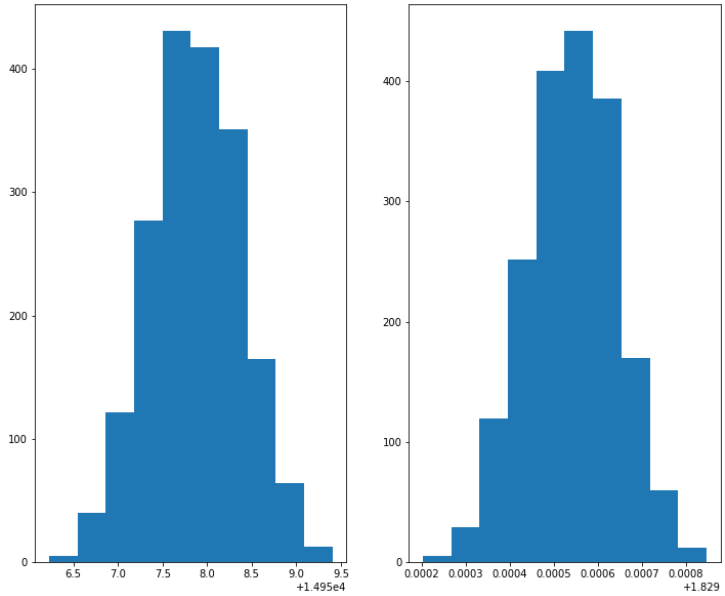


Figure 15. Histogram of the Parameters after Calibration and Burn-In.

```
print(mean_calibration)
[1.49578378e+04 1.82953523e+00]
```

Validation

accepted_validation, rejected_validation = metropolis_hastings(log_like, posterior_new_prior, transition_model_validation , prior_new, 100000, data_validation, acceptance_rule)

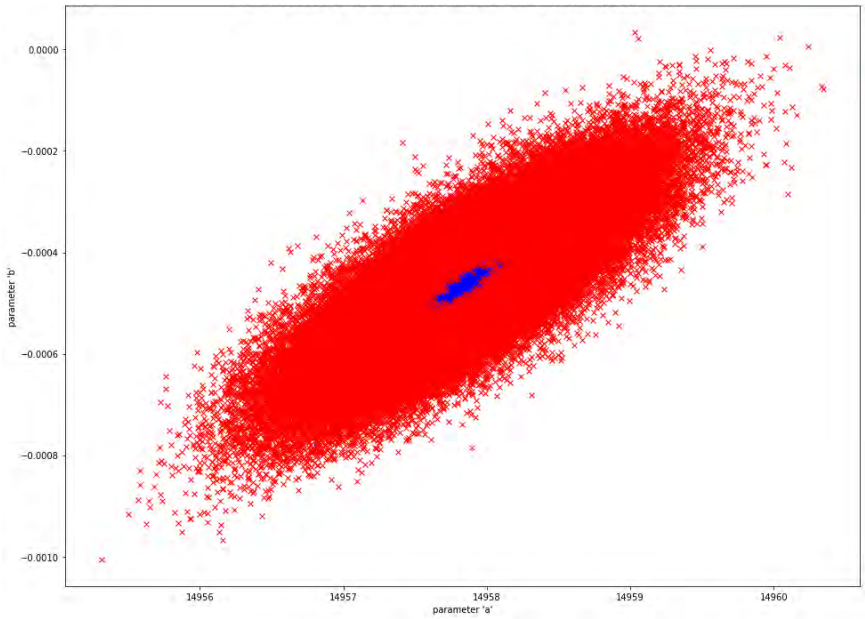


Figure 16. MCMC Sampling for Parameters of Testing Data. All Samples Are Shown.

Conclusion: Fitting Curves and QoI

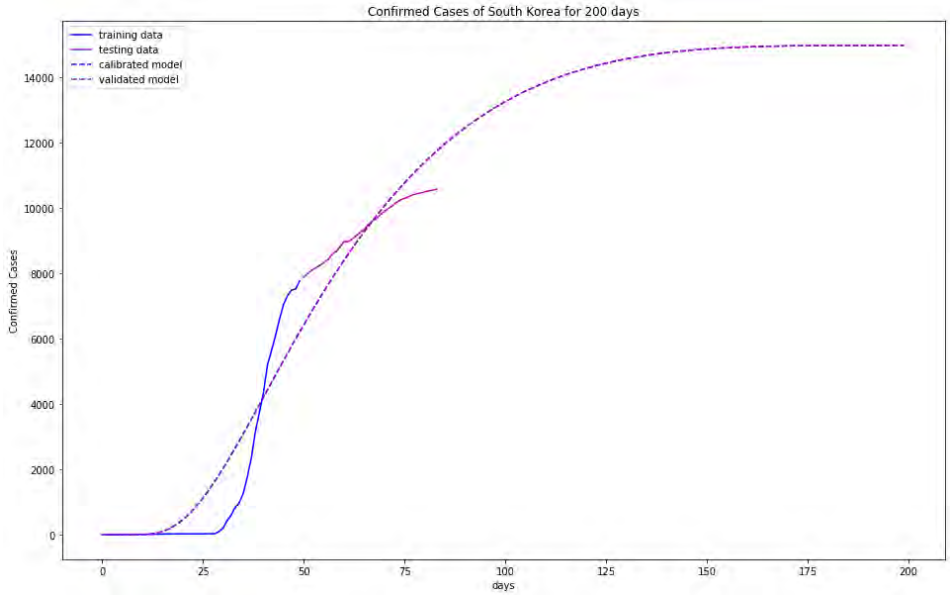


Figure 17. Curve Fitting from Training and Testing data Compared to the Observable Data.

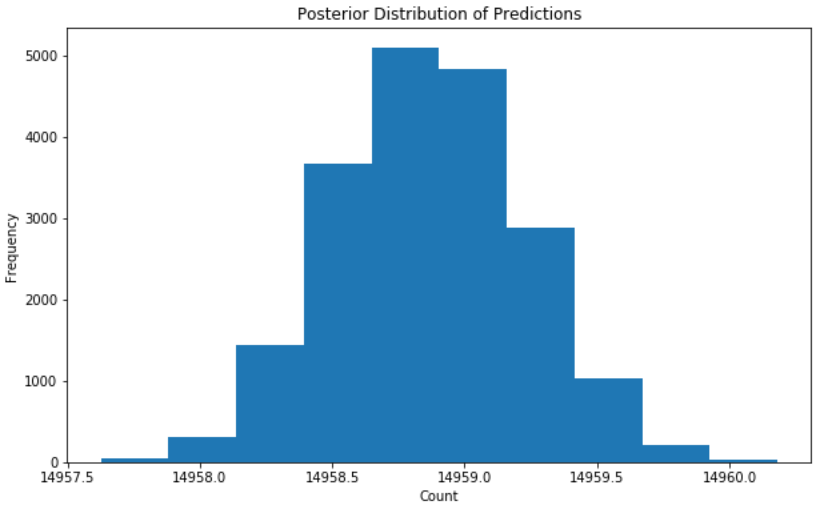
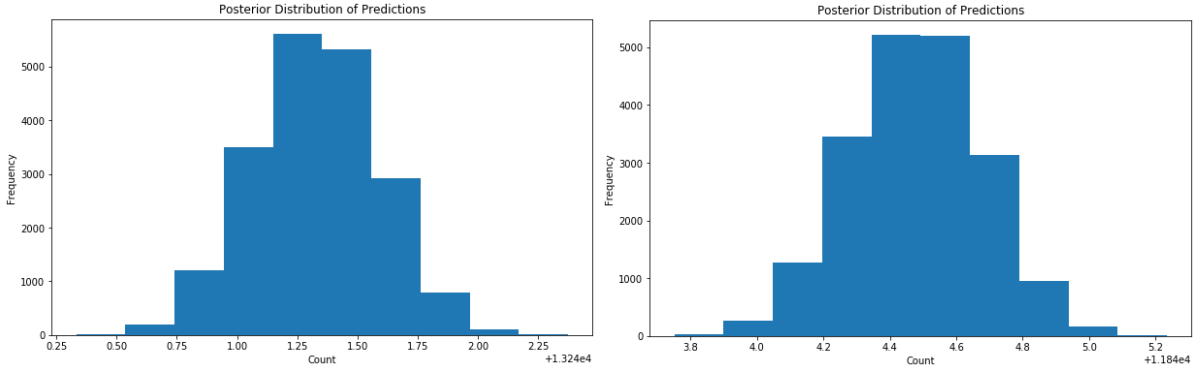


Figure 18. Posterior Distribution of Predictions at the 200th Day.



a. **b.**

Figure 15. Posterior Distribution of Predictions at the (a) 100th Day (b) 84th Day.

Since the evidence may not be sufficient enough to predict to the 200th day, the discrepancy is much larger. The error at the 84th day is around 10% which is 10 times greater than the prediction of 100 days. The results may not be sufficiently reliable.

Reference

1. J. Moukarzel., “From Scratch: Bayesian Inference, Markov Chain Monte Carlo and Metropolis Hastings, in python.”, Nov 2015 https://towardsdatascience.com/from-scratch-bayesian-inference-markov-chain-monte-carlo-and-metropolis-hastings-in-python-ef21a29e25a
2. K.V. Yuen, J. Beck, L.Katafygiotis, “Efficient Model Updating and Health Monitoring Methodology Using Incomplete Modal Data Without Mode Matching”, Struct. Control Health Monitor. 13 (2006) 91-107
3. **Viboud et al 2015** Viboud, C., Simonsen, L. and Chowell, G., 2016. A generalized-growth model to characterize the early ascending phase of infectious disease outbreaks. Epidemics, 15, pp.27-37.