STAT 840 final project

Since my background is Computational Mathematics, and the course is about computation inference, which based on my understanding, is exchanging complex mathematics with computing power and generality, I will try to use the most intuitive method based on my understanding of the model, while avoiding catastraphically terrible computing issues (i.e., direct operations and storage of an n-by-n dense matrix).

The model is that, for any given past condition u_{i-1}, the next condition $u_i|u_{i-1} \sim N(u_{i-1}, \sigma^2)$ is dependent on the past condition through normal distribution. The current condition is the logit residual of the current event frequency λ_i, and the frequency decides the number of events that would occur on the current day through parameterizing a Poisson model.

Since this is a very classical hierarchical model setup, we may prefer using an EM algorithm or a Monte Carlo (MC) method to derive the MLE, which avoids cumbersome mathematical derivations. I decide to select MC rather than EM, because the posterior latent density $u|y$ over which we take the expectation $Q(\beta_i, \sigma_i|\beta_{i-1}, \sigma_{i-1})$ can be hard to obtain, when the latent follows a normal distribution, whereas the observed data follows a Poisson distribution.

The joint distribution of the latent variable

No matter which method we use, we need to refine the long conditional chain into a joint distribution of the latent variable $u = [u_1, ..., u_n]^T$. Note that an equivalent form of the latent variable model is $u_i - u_{i-1} = \epsilon_i \sim N(0, \sigma^2)$, we can see that this is a simple Gaussian Markov Random Field (GMRF). Thus, we can use a self-rowwise-difference matrix $D \in R^{n\times n}$ to convert u to $\epsilon \sim N(0, \sigma^2 I_n)$.

Since I gave very detailed proof of this in Assignment 1, I will only repeat the most important steps here:

Define $D = \begin{bmatrix} 1 & & & & \\ -1 & 1 & & & \\ & -1 & 1 & & \\ & & & \ddots & \\ & & & -1 & 1 \end{bmatrix} \in R^{n\times n}$, then

$$Du = \begin{bmatrix} u_1 \\ u_2 - u_1 \\ \vdots \\ u_n - u_{n-1} \end{bmatrix} = \epsilon \sim N(0, \sigma^2 I_n)$$

Since $\det D = 1$ by its lower-triangularity, D^{-1} exists by its full rank, so

$$u = D^{-1}Du = N(0, \sigma^2 D^{-1} I_n D^{-T}) = N(0, \sigma^2 (D^T D)^{-1})$$

, where the precision matrix $\frac{1}{\sigma^2}Q = D^T D$ is sparse indeed due to the conditional independence from the Markov property.

Therefore, the latent density function is:

$$f(u;\sigma) = (2\pi)^{-n/2} \det(\sigma^2 (D^T D)^{-1})^{-1/2} \exp\left(-\frac{1}{2} u^T \frac{1}{\sigma^2} D^T D u\right)$$

We can simplify the covariance determinant by:

$$|\sigma^2 (D^T D)^{-1}| = \sigma^{2n} \frac{1}{|D^T D|} = \sigma^{2n} \frac{1}{|D|^2} = \sigma^{2n}$$

$$\det(\sigma^2 (D^T D)^{-1})^{-1/2} = (\sigma^{2n})^{-1/2} = \sigma^{-n}$$

Also, we can simplify the quadratic form in the exponential term by:

$$-\frac{1}{2} u^T \frac{1}{\sigma^2} D^T D u = -\frac{1}{2\sigma^2} u^T D^T D u = -\frac{1}{2\sigma^2} \|Du\|_2^2$$

Hence, the simplified latent density function is:

$$f(u;\sigma) = (2\pi)^{-n/2} \sigma^{-n} \exp\left(-\frac{1}{2\sigma^2} \|Du\|_2^2\right)$$

An MC maximum likelihood algorithm

Now, since we have obtained the latent density function, and the conditional likelihood is fairly straightforward to calculate, we may try to approximate the marginal likelihood using MC for the integral, then optimize the log marginal likelihood over β_0, σ to get the MLE of them.

Here, we devise the closed-form conditional density first.

For any $i = 1, 2, ..., n$, $Y_i | u_i \sim \text{Poisson}(\lambda_i)$, where $\lambda_i = \exp(\beta_0 + u_i)$, so the conditional density of a single day y_i is:

$$f(y_i | u_i) = \frac{e^{-\lambda_i} \lambda_i^{y_i}}{y_i!} = \frac{e^{-\exp(\beta_0 + u_i)} \exp(y_i(\beta_0 + u_i))}{y_i!}$$

Since every day is independent, the conditional density of all the n days should be:

$$f(y | u; \beta_0) = \prod_{i=1}^{n} \frac{e^{-\exp(\beta_0 + u_i)} \exp(y_i(\beta_0 + u_i))}{y_i!}$$

Then, we have reached a possible MC approximation for the marginal density. We first sample as many latent $u \sim N(0, \sigma^2 (D^T D)^{-1})$ as possible, then we approximate the marginal density by using:

$$f(y) = \int f(y|u) f(u) du \approx \frac{1}{L} \sum_{l=1}^{L} f(y|u_l), \; u_1, ..., u_L \sim^{i.i.d} N(0, \sigma^2 (D^T D)^{-1})$$

The problem here is that the latent sampling is inconvenient. But recall that $Du \sim N(0, \sigma^2 I_n)$, and $z = \frac{1}{\sigma} Du \sim N(0, I_n)$, we can sample a series of standard normal random vectors for the approximation.

Now, we use the change-of-variable to rearrange the approximation:

First, we have the change-of-variable formula:

$$z = \frac{1}{\sigma} Du$$

$$Du = \sigma z$$

$$u = \sigma D^{-1} z$$

Hence, the rearranged latent density becomes:

$$f(u; \sigma) = (2\pi)^{-n/2} \sigma^{-n} \exp\left(-\frac{1}{2\sigma^2} \|\sigma z\|_2^2\right) = (2\pi)^{-n/2} \sigma^{-n} \exp\left(-\frac{1}{2} \|z\|_2^2\right)$$

Then, we need a Jacobian determinant to account for the change-of-variable distortion in the integral:

$$\frac{du}{dz} = \sigma D^{-1}$$

$$\left|\frac{du}{dz}\right| = \left|\sigma D^{-1}\right| = \sigma^n \frac{1}{|D|} = \sigma^n$$

Hence, the final integral becomes

$$f(y) = \int f(y|u) f(u) du = \int f(y|u) (2\pi)^{-n/2} \sigma^{-n} \exp\left(-\frac{1}{2} \|z\|_2^2\right) \sigma^n dz =$$

$$\int f(y|u) (2\pi)^{-n/2} \exp\left(-\frac{1}{2} \|z\|_2^2\right) dz = \int f(y|u) \left[(2\pi)^{-n/2} \exp\left(-\frac{1}{2} \|z\|_2^2\right)\right] =$$

$$\int f(y|u) \phi(z) dz$$

, where $\phi(z)$ is the probability density function (PDF) of a standard normal distribution $N(0, I_n)$.

Finally, we derive a much more convenient MC apprximation for the marginal lieklihood:

$$f(y) \approx \frac{1}{L} \sum_{l=1}^{L} f(y|\sigma D^{-1} z_l), \ z_1, ..., z_L \sim^{i.i.d} N(0, I_n)$$

, in terms of both density computing and latent sampling, where

$$f(y|\sigma D^{-1} z_l) = \prod_{i=1}^{n} \frac{e^{-\exp(\beta_0 + \sigma(D^{-1} z_l)_i)} \exp\left(y_i (\beta_0 + \sigma(D^{-1} z_l)_i)\right)}{y_i!}$$

, and the notation $(D^{-1} z_l)_i$ means the i 'th entry of the vector $D^{-1} z_l$ MC sample.

Note that the inverse of the difference matrix $D^{-1} \in R^{n \times n}$ might not be sparse or efficient to compute, we should find an alternative to get compute $D^{-1} z_l$ without having to store/compute

such a giant dense inverse matrix.

Let $v_l = D^{-1} z_l$, then v_l is the solution to this matrix equation: $D v_l = z_l$. The solution is unique since D is full-rank. The computing is very efficient now, because D is sparse, lower-triangular, and of band 1. We can easily store it in $O(n)$, and solve the equation to get $v_l = D^{-1} z_l$ in $O(n)$ using band-1 backward substitution.

Hence, we devise such an expression for the PDF of $y|u$:

$$f(y|\sigma D^{-1} z_l) = \prod_{i=1}^{n} \frac{e^{-\exp(\beta_0 + \sigma v_{li})} \exp\left(y_i(\beta_0 + \sigma v_{li})\right)}{y_i!}$$

, where $v_l = D^{-1} z_l$.

We can see that the algorithm has a trivial MC sampling from the standard normal distribution, and a very efficient $O(n)$ storage/computing.

What's more, in MC approximation, the machine epsilon problem plays an important role, so we would better rescale the numbers to a more common interval for operation:

First, we rescale the inner conditional PDF product to log so that we change the multiplication of the small numbers into additions:

$$\log f(y|\sigma D^{-1} z_l) = \sum_{i=1}^{n} \log\left(\frac{e^{-\exp(\beta_0 + \sigma v_{li})} \exp\left(y_i(\beta_0 + \sigma v_{li})\right)}{y_i!}\right) =$$

$$\sum_{i=1}^{n} \left[\log(e^{-\exp(\beta_0 + \sigma v_{li})}) + \log \exp\left(y_i(\beta_0 + \sigma v_{li})\right) - \log y_i!\right] =$$

$$\sum_{i=1}^{n} \left[-\exp(\beta_0 + \sigma v_{li}) + y_i(\beta_0 + \sigma v_{li}) - \log y_i!\right] = -\sum_{i=1}^{n} e^{\beta_0 + \sigma v_{li}} + \sum_{i=1}^{n} y_i(\beta_0 + \sigma v_{li}) -$$

$$\sum_{i=1}^{n} \log y_i!$$

Then, we define the numerically stable likelihood function to be:

$$L(\beta_0, \sigma) = -\log f(y) \approx -\log\left(\frac{1}{L}\sum_{l=1}^{L} f(y|\sigma D^{-1} z_l)\right) = \log L - \log \sum_{l=1}^{L} f(y|\sigma D^{-1} z_l) =$$

$$\log L - \log \sum_{l=1}^{L} \exp\left(\log\left(f(y|\sigma D^{-1} z_l)\right)\right) = \log L - \mathrm{logSumExp}\left(\log\left(f(y|\sigma D^{-1} z_l)\right)\right)$$

, where

$$\log f(y|\sigma D^{-1} z_l) = -\sum_{i=1}^{n} e^{\beta_0 + \sigma v_{li}} + \sum_{i=1}^{n} y_i(\beta_0 + \sigma v_{li}) - \sum_{i=1}^{n} \log y_i!$$

Now, our MC algorithm is easy to sample (from a trivial standard normal distribution) by using a change-of-variable, efficient to store and compute by taking advantage of the sparsity and other structures of the precision matrix, and numerically stable by avoiding multiplying a series of very small numbers.

Then, to get a fast optimization algorithm for the MLE, we at least need to derive the gradient of the marginal NLL. Fortunately, a closed form of the MC approximated marginal NLL seems tractable and not too complicated.

To use chain rule for the gradient, we start from the most inner derivative:

Define $k_l(\beta_0, \sigma) = \log f(y|\sigma D^{-1} z_l) = -\sum_{i=1}^{n} e^{\beta_0 + \sigma v_{li}} + \sum_{i=1}^{n} y_i(\beta_0 + \sigma v_{li}) - \sum_{i=1}^{n} \log y_i!$, we have

$$\frac{\partial k_l}{\partial \beta_0} = -\sum_{i=1}^{n} e^{\beta_0 + \sigma v_{li}} + \sum_{i=1}^{n} y_i$$

, and

$$\frac{\partial k_l}{\partial \sigma} = -\sum_{i=1}^{n} v_{li} e^{\beta_0 + \sigma v_{li}} + \sum_{i=1}^{n} y_i v_{li}$$

Hence, we rewrite the marginal NLL as

$L(\beta_0, \sigma) = \log L - \text{logSumExp}\left(\log\left(f(y|\sigma D^{-1} z_l)\right)\right) = \log L - \log\left(\sum_{i=1}^{l} e^{k_l}\right)$, so the gradient can be derived:

$$\frac{\partial L}{\partial \beta_0} = -\frac{1}{\sum_{i=1}^{l} e^{k_l}} \sum_{i=1}^{l} \frac{\partial e^{k_l}}{\partial k_l} \frac{\partial k_l}{\partial \beta_0} = -\frac{1}{\sum_{i=1}^{l} e^{k_l}} \sum_{i=1}^{l} e^{k_l} \left[-\sum_{i=1}^{n} e^{\beta_0 + \sigma v_{li}} + \sum_{i=1}^{n} y_i\right] = \frac{1}{\sum_{i=1}^{l} e^{k_l}} \sum_{i=1}^{l} e^{k_l} \left[\sum_{i=1}^{n} e^{\beta_0 + \sigma v_{li}} - \sum_{i=1}^{n} y_i\right]$$

$$\frac{\partial L}{\partial \sigma} = -\frac{1}{\sum_{i=1}^{l} e^{k_l}} \sum_{i=1}^{l} \frac{\partial e^{k_l}}{\partial k_l} \frac{\partial k_l}{\partial \sigma} = -\frac{1}{\sum_{i=1}^{l} e^{k_l}} \sum_{i=1}^{l} e^{k_l} \left[-\sum_{i=1}^{n} v_{li} e^{\beta_0 + \sigma v_{li}} + \sum_{i=1}^{n} y_i v_{li}\right] = \frac{1}{\sum_{i=1}^{l} e^{k_l}} \sum_{i=1}^{l} e^{k_l} \left[\sum_{i=1}^{n} v_{li} e^{\beta_0 + \sigma v_{li}} - \sum_{i=1}^{n} y_i v_{li}\right]$$

Thus, the gradient of the MC approximated marginal NLL is:

$$\nabla_{[\beta_0, \sigma]^T} L = \frac{1}{\sum_{i=1}^{l} e^{k_l}} \begin{bmatrix} \sum_{i=1}^{l} e^{k_l} \left[\sum_{i=1}^{n} e^{\beta_0 + \sigma v_{li}} - \sum_{i=1}^{n} y_i\right] \\ \sum_{i=1}^{l} e^{k_l} \left[\sum_{i=1}^{n} v_{li} e^{\beta_0 + \sigma v_{li}} - \sum_{i=1}^{n} y_i v_{li}\right] \end{bmatrix}$$

, where $k_l = -\sum_{i=1}^{n} e^{\beta_0 + \sigma v_{li}} + \sum_{i=1}^{n} y_i(\beta_0 + \sigma v_{li}) - \sum_{i=1}^{n} \log y_i!$

However, the devised algorithm suffers severely from the machine epsilon problem. Unlike likelihood functions, we can solve the problem by rescaling them to log scale while keeping the MLE the same based on the increasing property of log functions. The gradient itself can't be converted to its log without affecting the equivalence, so I give up using a more efficient gradient here. Also, building an algorithmic differentiation function for the gradient requires highly customized algorithm design and implementation, which is time consuming, which should be considered only when a numerical one doesn't have enough accuracy. Indeed, we

don't lose too much accuracy by numerically approximating first derivatives, so the gradient is eventually decided to be computed by numerical algorithms.

Another decision to make it whether to use Hessian matrices. We can take a look at the general shape of the negative log-marginal likelihood to guide our initial value selection:

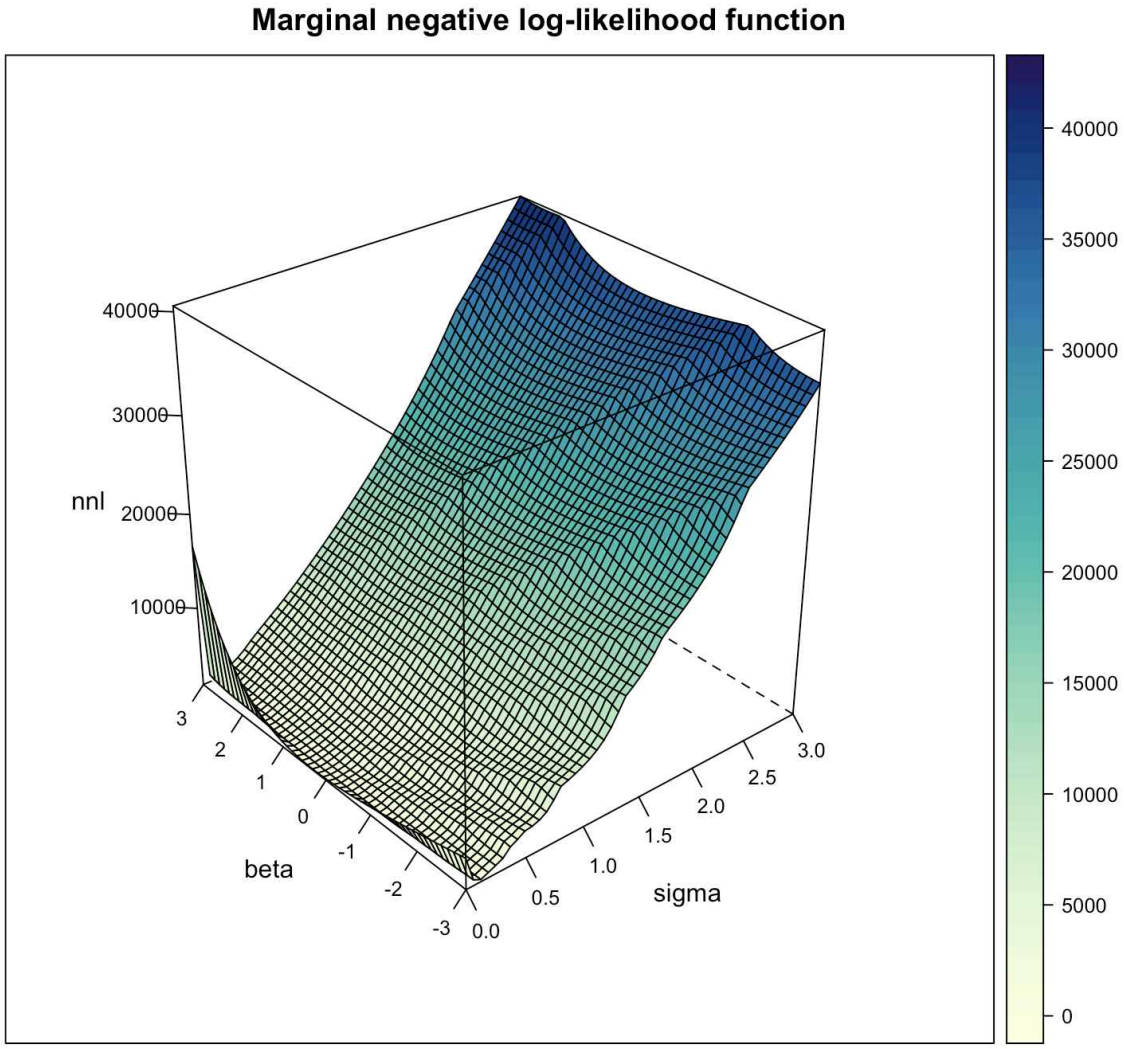

Fig.1 Parameters vs. the negative log-marginal likelihood

The plot in Fig.1 is generated using an MC of sample size 800 to make sure the shape is fairly stable. The plot doesn't change dramatically as I further increase the sample size, which is a reason why I choose this sample size. Other performance for the sample size is also tested below.

The marginal NLL seems non-convex, and not even quasi-convex, which hinders us from computing the MLE using Newton's method. The algorithm will be dangerous with its non-quasi-convexity, even with measures like direction correction or step size halving. Therefore, we conclude that Newton's method does not fit the NLL.

At the end of this section, we derived and designed an intuitive and efficient algorithm for calculating the marginal NLL. We also discussed and determined a suitable strategy for optimizing it for the MLE.

MCMLE results and confidence intervals

After implementing the marginal NLL algorithm that generates an MC approximation in Fig.1, we directly use numerically approximated gradient descent algorithm to get the MLE. By setting an MC sample size of 800, I got an MLE of $\hat{\beta}_0 = 0.19603344, \hat{\sigma} = 0.06846885$. The initial values I selected for β_0 is $[-1, 1]$, and $[0.1, 1]$ for σ. The selection is based on the marginal NLL plot in Fig.1, where the region seems to be the lowest part and nearly locally convex. The MC sample size is a balance between time consumption and result stability. Since the MC samples are of very high dimension (of 1000), we should avoid sampling too many even though it's just a simple standard normal distribution. The stability for the sample size choice is analyzed by the MC-sampling bootstrapping confidence interval later.

To test its sensitivity to different initial values, we try 100 grids of the initial values from the reasonable region $(\beta_0, \sigma) \in [-1, 1] \times [0.01, 1]$ with a fixed set of MC samples and see how many times it converges to a reliable MLE. Since the computation is time-consuming, I decrease the MC sample size from 800 to 300. Even though the MC approximation of the marginal NLL is expected to be much worse, the algorithm can be more sufficiently proven to be robust against initial values if it doesn't yield very unreliable results:

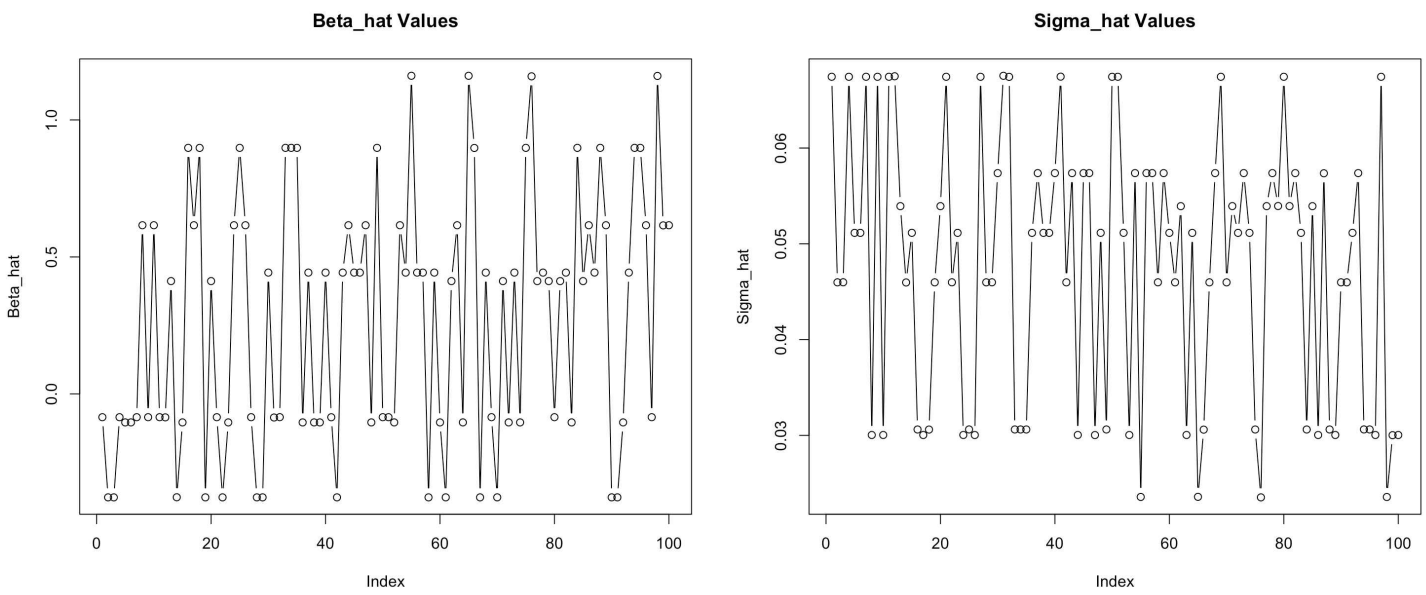

Fig.2 Converged MLE for initial value grids betweem [-1, 1]X[0.01, 1]

According to Fig.2, we see that as long as the initial value is within the reasonable region $(\beta_0, \sigma) \in [-1, 1] \times [0.01, 1]$, the MLE never breaks the boundary of the region too far, indicating a relatively robust performance against initial values.

To quantify the uncertainty of the MLE, we derive its 95% confidence interval. Since it's not clear which one method is the most suitable for computing a confidence interval for an MCMLE like this, I will try multiple methods, and discuss their expected pros and cons respectively.

First, based on the method we selected, we notice that the MLE's randomness mainly comes from 2 sources: MC sampling, and observational data sampling. I believe in my method, the MC sampling might contribute more to the variance of the MLE, but I will quantify both.

To quantify the randomness from MC sampling, we first notice that the MC samples are from the standard normal distribution $N(0, I_n)$. Since the sample distribution is clear, and the MLE is just a function of the MC samples, we can, while holding the observational data fixed, use parametric bootstrapping to get a 95% confidence interval.

I decided to re-sample MC samples $B_1 = \{z_1^1, ..., z_{800}^1 \sim N(0, I_n)\}, ..., B_{300}$ 300 times, each time we sample 800 vectors from the standard normal distribution and get an MLE $(\hat{\beta}_0^1, \hat{\sigma}^1), ..., (\hat{\beta}_0^{300}, \hat{\sigma}^{300})$ based on the MC sample. The number of MC samples is determined by trials, when I found that the interval doesn't change dramatically when I increase the size. Finally, we draw the empirical 95% quantile around the sample mean MLE (i.e., 2.5% empirical quantile ~ 97.5% empirical quantile, same for the second method later) to have the confidence interval. This method is very intuitive, free of math, and has the minimum assumption, thus more interpretable and preferable. Other methods, such as taking the inverse of the Hessian matrix of the marginal NLL, are not considered, because 1) It's not as intuitive and general (e.g., free of assumptions) as this one, and 2) Computing the Hessian matrix can be challenging: I failed to derive a numerically stable closed-form, designing an AD algorithm can be time consuming, and numerically approximating second-order derivatives is always risky.

As a result, the 95% confidence interval for $\hat{\beta}_0$ is $[-0.271498, 1.020979]$ with a mean of 0.2643378. The empirical 95% confidence interval for $\hat{\sigma}$ is $[0.03139594, 0.08105699]$ with a mean of 0.05369109. The MLE for $\hat{\beta}_0$ seems a little variational regarding the MC sampling, and the other parameter seems just fine. The reason and more analysis will be given when we get all the results.

Then, to quantify the randomness of observational data sampling, we must use a parametric bootstrapping. This is because if we use nonparametric bootrapping and resample the original dataset, we ruin the time-dependence structure of the data. To preserve the time dependence, we must start from the bottom latent variable. Hence, we sample the dataset from $\text{Poisson}(\lambda_i)$ parameterized by the latent prediction $\hat{u}$. The latent prediction is obtained in the following section, which we will cover later.

After implementing and executing the bootstrapping, the 95% confidence interval for $\hat{\beta}_0$ is $[-0.01243307, 0.79187909]$ with a mean of 0.1893628. The empirical 95% confidence interval for $\hat{\sigma}$ is $[0.03592138, 0.08028987]$ with a mean of 0.05667261. The conclusion doesn't change too much. Also, just as what we expected, the confidence interval significantly shrinks compared to the previous one, indicating the randomness of the MC algorithm itself is more significant than the dataset sampling.

Based on the result, we can see that the MLE for σ is very stable, where the MLE for β_0 is very variational. Indeed, according to Fig.1, the marginal NLL changes rapidly w.r.t. σ, but very flat w.r.t. β_0, especially between the interval $[-1, 1]$. Hence, a tiny change in $\hat{\sigma}$ yields a very different NLL value, but a fairly large change in $\hat{\beta}_0$ around 0 doesn't really change the NLL value much, which makes it "less identifiable" than the other parameter, i.e., more variational. In

another aspect, the gradients around the very flat region are all close-to-0, which makes them all good candidates for the MLE.

Latent variable prediction algorithm

After getting the MLE, we can parameterize the hierarchical model and obtain a set of predictions for the latent variable $\hat{u} = \arg\max P(u; y, \hat{\sigma}, \hat{\beta}_0)$. We will find a way to compute the prediction, as well as its 95% confidence interval.

To solve the latent maximum likelihood problem for its prediction, we can first derive its gradient, which enables gradient descent, and is necessary if we can even use some better methods, such as Newton's method or analytical solution.

By Bayesian rule, we have the conditional (posterial) latent likelihood:

$$P(u; y, \hat{\sigma}, \hat{\beta}_0) = \log f(y|u; \hat{\beta}_0) + \log f(u; \hat{\sigma})$$

We plug in $\log f(y|u; \hat{\beta}_0)$ we obtained in the last section to get:

$$P(u; y, \hat{\sigma}, \hat{\beta}_0) = -\sum_{i=1}^{n} e^{\hat{\beta}_0 + u_i} + \sum_{i=1}^{n} y_i(\hat{\beta}_0 + u_i) - \sum_{i=1}^{n} \log y_i! + \log f(u; \hat{\sigma})$$

Then, we simplify the last term, which is the latent log-likelihood:

$$\log f(u; \hat{\sigma}) = \log \left[(2\pi)^{-n/2} \hat{\sigma}^{-n} \exp\left(-\frac{1}{2\hat{\sigma}^2} \|Du\|_2^2 \right) \right] = -\frac{n}{2} \log(2\pi) - n \log(n) - \frac{1}{2\hat{\sigma}^2} \|Du\|_2^2$$

Thus, the conditional latent likelihood becomes:

$$P(u; y, \hat{\sigma}, \hat{\beta}_0) = -\sum_{i=1}^{n} e^{\hat{\beta}_0 + u_i} + \sum_{i=1}^{n} y_i(\hat{\beta}_0 + u_i) - \sum_{i=1}^{n} \log y_i! - \frac{n}{2} \log(2\pi) - n \log(n) - \frac{1}{2\hat{\sigma}^2} \|Du\|_2^2$$

By removing all the terms irrelevant to the latent variable (i.e., constant w.r.t. the latent variable), we have

$$P(u; y, \hat{\sigma}, \hat{\beta}_0) \equiv -\sum_{i=1}^{n} e^{\hat{\beta}_0 + u_i} + \sum_{i=1}^{n} y_i(\hat{\beta}_0 + u_i) - \frac{1}{2\hat{\sigma}^2} \|Du\|_2^2$$

Or, preferably, we write the likelihood as a negative log-likelihood (NLL):

$$L(u; y, \hat{\sigma}, \hat{\beta}_0) = \sum_{i=1}^{n} e^{\hat{\beta}_0 + u_i} - \sum_{i=1}^{n} y_i(\hat{\beta}_0 + u_i) + \frac{1}{2\hat{\sigma}^2} \|Du\|_2^2$$

To take the gradient of the NLL w.r.t. a gradient, we prefer a vector-only form:

$$L(u; y, \hat{\sigma}, \hat{\beta}_0) = e^{\hat{\beta}_0} 1^T \begin{bmatrix} e^{u_1} \\ \vdots \\ e^{u_n} \end{bmatrix} - \hat{\beta}_0 1^T y - y^T u + \frac{1}{2\hat{\sigma}^2} \|Du\|_2^2$$

Again, we remove the irrelevant term:

$$L(u; y, \hat{\sigma}, \hat{\beta}_0) \equiv e^{\hat{\beta}_0} 1^T \begin{bmatrix} e^{u_1} \\ \vdots \\ e^{u_n} \end{bmatrix} - y^T u + \frac{1}{2\hat{\sigma}^2} \|Du\|_2^2$$

Now, we are ready to derive the gradient w.r.t. the latent variable:

$$\nabla_u L = e^{\hat{\beta}_0} \begin{bmatrix} e^{u_1} \\ \vdots \\ e^{u_n} \end{bmatrix} - y + \frac{1}{2\hat{\sigma}^2} 2D^T Du = e^{\hat{\beta}_0} \begin{bmatrix} e^{u_1} \\ \vdots \\ e^{u_n} \end{bmatrix} - y + \frac{1}{\hat{\sigma}^2} D^T Du$$

For convenience, we denote $e^u = [e^{u_1}, ..., e^{u_n}]^T$, then the gradient is:

$$\nabla_u L = e^{\hat{\beta}_0} e^u - y + \frac{1}{\hat{\sigma}^2} D^T Du$$

Since the gradient contains a nonlinear transformation of u , which makes it not as easy to solve as a linear equation, we use a numerical algorithm instead. To further save memory storage and avoid inefficient computing, we replace the cross product matrix by the band-1 sparse matrix we defined at the beginning:

$$\nabla_u L = e^{\hat{\beta}_0} e^u - y + \frac{1}{\hat{\sigma}^2} Qu, \text{ where } Q = D^T D$$

, and we know that the storage and computing only requires $O(n)$.

Even though we can optimize the NLL using gradient descent now, the Hessian matrix seems trivial to obtain, and is beneficial since it enables Newton's method that is fast and has nice statistical properties. Additionally, we need its inverse to get the confidence interval for the prediction. Thus, we get the Hessian also:

$$H(u) = e^{\hat{\beta}_0} \text{diag}(e^{u_1}, ..., e^{u_n}) + \frac{1}{\hat{\sigma}^2} Q$$

Nonetheless, the computing of the inverse that is both required by Newton's method and the 95% confidence interval for the latent prediction is still a problem, since it is not sparse. We recall the strategy we applied in Assignment 1, and solve the problem by precomputing a Cholesky decomposition:

Let $H(u) = LL^T$ be the Cholesky decomposition of the Hessian matrix. The decomposition is done in-place and preserves the sparsity of the Hessian matrix, which takes $O(n)$ to compute and store here since $H(u)$ is of band 1.

Then, define $L^{-1} = [L_1^{-1}, ..., L_n^{-1}]$, where $L_1^{-1}, ..., L_n^{-1} \in R^n$ are the columns of the dense inverse L^{-1} , we have

$$LL^{-1} = I_n$$

$$L[L_1^{-1}, ..., L_n^{-1}] = [e_1, ..., e_n]$$

$$[LL_1^{-1}, ..., LL_n^{-1}] = [e_1, ..., e_n]$$

Hence,

$LL_i^{-1} = e_i, \forall i = 1, 2, ..., n$

Since L is full-rank, L_i^{-1} is the only solution to the linear equation $LL_i^{-1} = e_i$, and the computation and the storage takes $O(n)$ by band backward substitution since L is of band 1.

Then, the diagonal entry of the inverse Hessian is:

$H(u)^{-1} = L^{-T}L^{-1}$

$[H(u)^{-1}]_{ii} = (L^{-T}L^{-1})_{ii} = \|L_i^{-1}\|_2^2$

Thus, we derive a much more efficient way of getting the standard error of the latent variable:

$se(\hat{u}_i) = \sqrt{[H(u)^{-1}]_{ii}} = \|L_i^{-1}\|_2$

, where L_i^{-1} is the solution to the linear equation $LL_i^{-1} = e_i,\ i = 1, 2, ..., n$ that can be efficiently solved and stored by $O(n)$.

The computation and storage of the Hessian matrix and the gradient only takes $O(n)$ since Q and $\mathrm{diag}(e^u)$ is sparse, hence Newton's method should be very efficient.

Now, we need another way to fit the inverse Hessian information to Newton's method, which takes advantage of the second-order information of NLL, e.g., adapts a large step size when the gradient is steep.

We notice that when the NLL is non-convex, we need to correct the update direction by eigendecomposing the Hessian, turn the diagonal matrix's entries to one-over its absolute value, then combine them back to form the "absolution/positive" Hessian inverse. This operation is necessary when the non-convex NLL's Hessian inverse sometimes will lead the iterations to some opposite direction. If we indeed have to do this, we shouldn't have used Newton's method here since I don't know any eigendecomposition methods that can exploit the band-1 structure of the Hessian matrix to avoid $O(n^3)$ computing.

Fortunately, we can actually prove that the NLL is actually globally convex by showing its Hessian matrix is PSD:

First, we notice that $e^{\hat{\beta}_0}\mathrm{diag}(e^{u_1}, ..., e^{u_n})$ is always PSD since its entries are always positive for any u. Then, it suffices to show that Q is PSD, because $\frac{1}{\sigma^2}$ must be non-negative as well.

Indeed, for any $v \in R^n$,

$v^TQv = v^TD^TDv = (Dv)^T(Dv) = \|Dv\|_2^2 \geq 0$

Hence, Q is PSD, so $H(u) = e^{\hat{\beta}_0}\mathrm{diag}(e^{u_1}, ..., e^{u_n}) + \frac{1}{\hat{\sigma}^2}Q$ is PSD since the sum of 2 PSD matrices is also PSD. Furthermore, since the first matrix is PD, we can conclude that the Hessian matrix is PD.

Now that we already show the vanilla version of Newton's method works on the problem, we just find an efficient way of computing the update step in each iteration:

Note that the update step for any iteration is

$w_i = H(u_i)^{-1} g_i$, where $g_i = \nabla_u L(u_i)$, we have

$H(u_i) w_i = g_i$

Thus, we can solve the trivial linear equation $H(u_i) w_i = g_i$ by using band-1 backward substitution to get the update step vector w_i . Again, the solution is unique because $H(u_i)$ is PD and thus full-rank. Then, we just update the parameter by

$u_{i+1} \leftarrow u_i - w_i$

Now, we have completed all the algorithms to compute the latent predictions and their confidence intervals. The algorithms have been proved to converge and take only $O(n)$ for memory storing and computing.

Latent prediction and confidence interval results

After implementing the algorithms we derived in the previous section in R, we show the prediction results and their 95% confidence intervals.

The latent predictions are computed using Newton's method I implemented with the closed-form gradient and Hessian matrix functions. The initial latent u was set to be $u_0 = \beta_0 1$, and the stopping criterion is when the 2-norm of the gradient decreases to smaller than 1e-5. The convergence plot is shown in Fig.3:

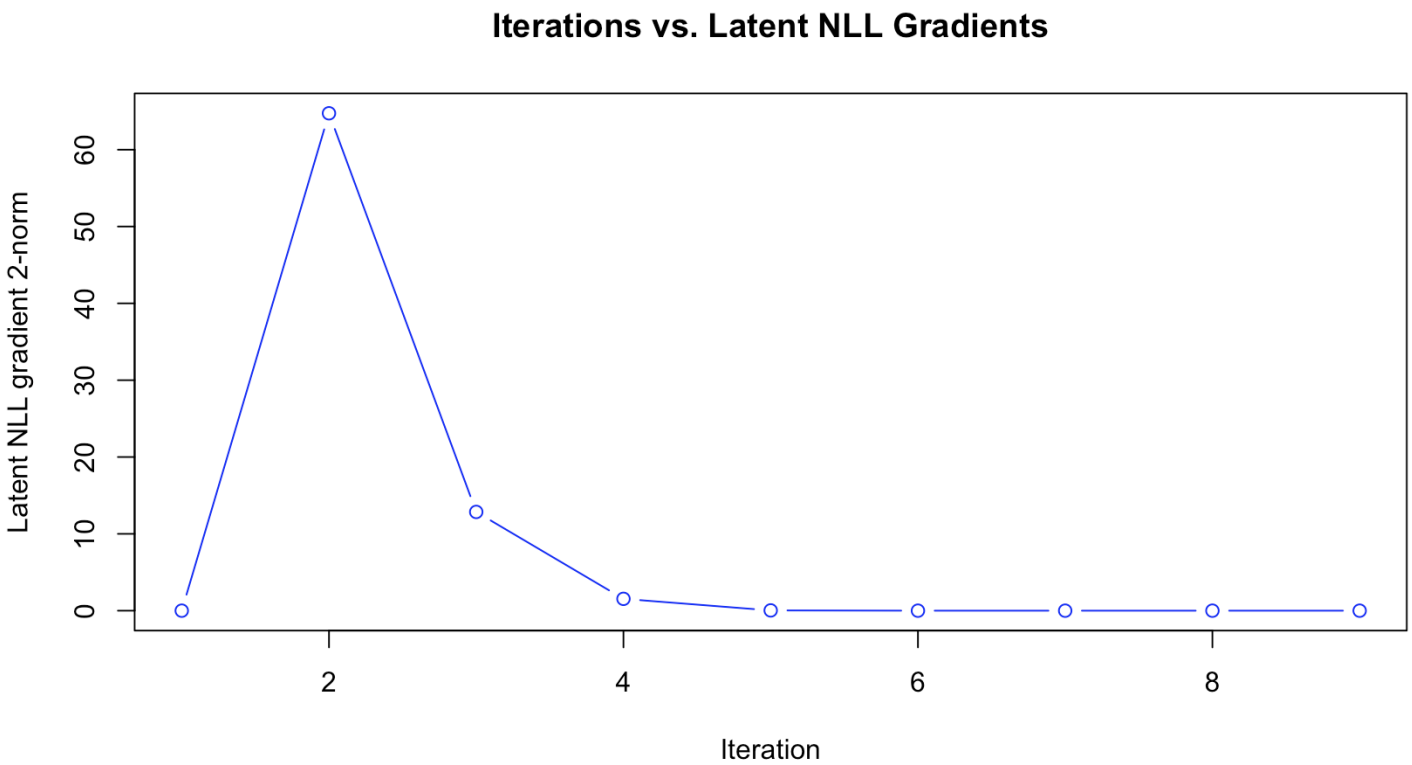

Fig.3 Iteration vs. Latent NLL gradient 2-norm

The algorithm takes around 9 iterations to converge and is equivalent to around 5 seconds.

The predictions and confidence intervals are of very high dimensions (of 1000), which is impossible to plot. Hence, according to our optimization lab, we use $\hat{u}$ as known to parameterize the Poisson model $\text{Poisson}(\exp(\beta_0 + u_i))$ for each $i = 1, 2, ..., n$. Then, we take $\hat{y}_i$ to be the expectation of the distribution, i.e., $\lambda_i = \exp(\beta_0 + u_i)$, and the confidence intervals follow the same idea.

Then, the prediction and confidence interval of the latent variable against the true data is shown in Fig.4:

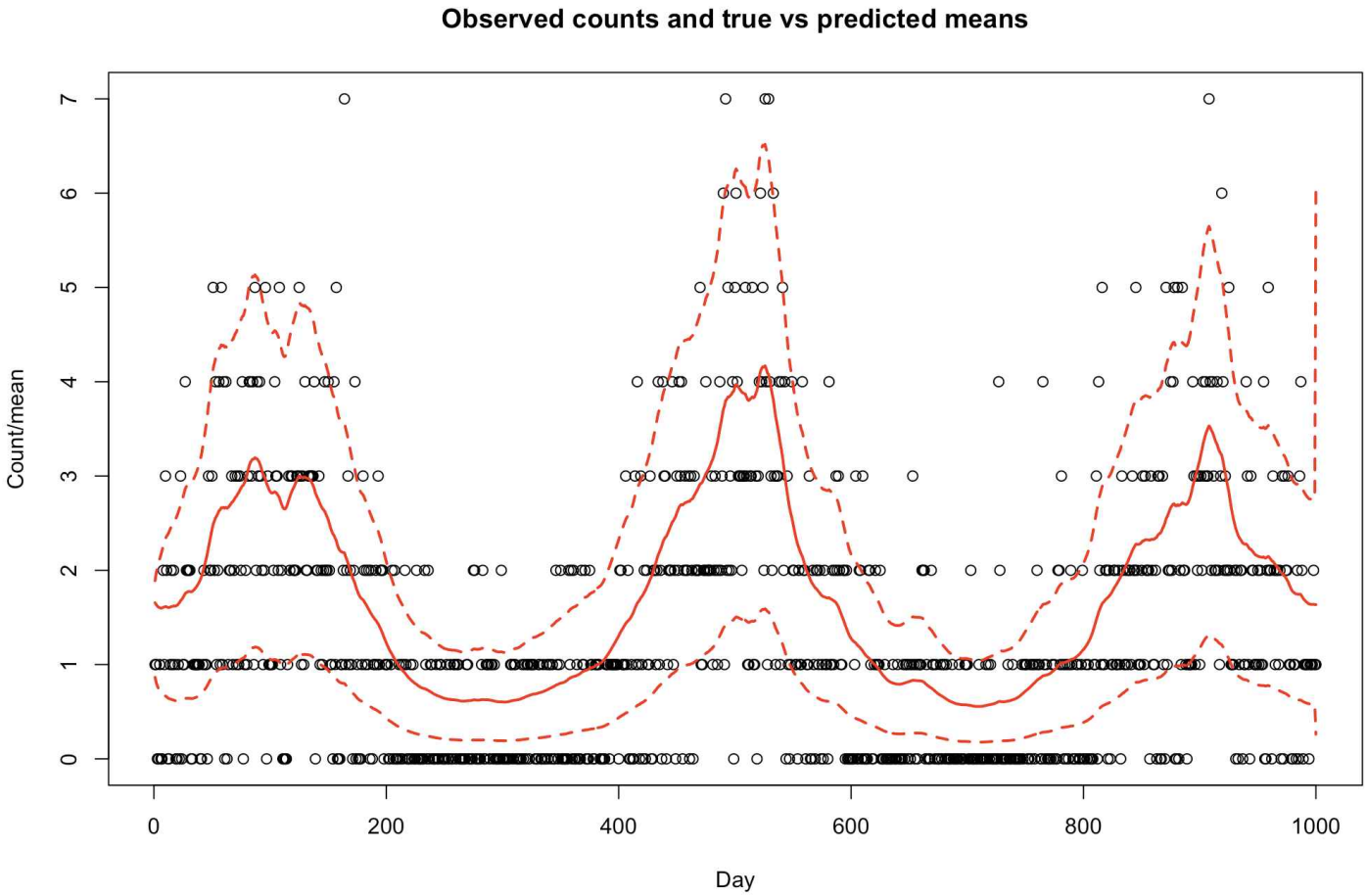

Fig.4 Prediction and 95% confidence interval for the latent parameterized mean count vs. true count

According to Fig.4, the predictions captures the overall trend of the time-series relationship well, and 95% confidence intervals are informative The results are appropriate, which means that we are very likely to have very close MLE from the series of computational inferences. The computations and storages are very fast, which takes seconds to complete.

Summary

Since the project is very comprehensive, and a lot of details are discussed, I summarize the key points here. Other discussions in the main body of the report, such as numerical stability analysis/tricks, are not considered the most important points, so are not covered here.

First, MCML is selected to be the method for computing the MLE, because 1) the analytical marginal density is very hard to derive using math, 2) MCML fits such a hierarchical model setup

fairly well, and 3) an EM algorithm is harder here, because the conditional (posterior) latent density is challenging to derive.

Then, after approximating the marginal NLL using MC, we obtain a function of shape shown in Fig.1. The function is neither convex nor quasi-convex, so we can use up to the first order derivative to optimize it, i.e., Newton's method is not applicable safely here. Also, the analytical gradient of the NLL is, even though derived by me, suffering from numerical problems as it involves multiplications of a series of very tiny numbers. Thus, since the numerical approximations of first-order derivatives usually have acceptable accuracies, I used a descent gradient method with sufficiently small step size and a numerically approximated NLL gradient. The results are the convergence plot shown in Fig.3, and the MLE is $\hat{\beta}_0 = 0.19603344, \hat{\sigma} = 0.06846885$.

Two types of confidence intervals are computed for the MLEs, with each emphasizing one of the sampling randomness from the MC algorithm itself, or the observational data. A parametric bootstrapping is used for quantifying the observational dataset sampling randomness impacts, since otherwise, a nonparametic bootstrpping breaks the time dependence strucuture of the dataset, and thus will eliminate the σ. The first 95% confidence interval for $\hat{\beta}_0$ is $[-0.271498, 1.020979]$ with a mean of 0.2643378. The empirical 95% confidence interval for $\hat{\sigma}$ is $[0.03139594, 0.08105699]$ with a mean of 0.05369109. The second 95% confidence interval for $\hat{\beta}_0$ is $[-0.01243307, 0.79187909]$ with a mean of 0.1893628. The empirical 95% confidence interval for $\hat{\sigma}$ is $[0.03592138, 0.08028987]$ with a mean of 0.05667261. According to the results, $\hat{\beta}_0$ has a much more variational confidence interval than $\hat{\sigma}$, because the NLL is very flat w.r.t. σ around $[-1, 1]$, which makes the parameter "less identifiable" than the other one. Also, the confidence interval accounting for the MC sampling is significantly wider than the one for the dataset sampling itself, indicating that the MC sampling process might contribute more randomness than the dataset sampling.

Then, we use Bayesian rule to derive the complete-data NLL, based on which we design a series of algorithms to find latent predictions and confidence intervals. The NLL is computed efficiently in $O(n)$ in terms of both memory storage and time by using Cholesky decomposition and exploiting the band-1 and sparsity structures of the precision/Hessian matrices. The analytical form of the gradient and Hessian are derived and implemented to get both the NLL and the confidence interval. Newton's method is used to find the latent predictions, which is considered to be optimal here, because 1) the NLL is proved to be globally convex, and 2) the gradient and Hessian are computed using analytical form in a very efficient and accurate way. The convergence plot for the prediction is shown in Fig.3, which takes 9 iterations to converge to a tolerance of 1e-5. The algorithm is indeed stable, robust, accurate, and efficient.

The confidence intervals are obtained by computing the diagonal entries of the inverse Hessian at the prediction points. A Cholesky decomposition and band backward substitution are also applied here to efficiently avoid processing the dense inverse Hessian matrices. The confidence

interval is shown in Fig.4. The overall latent prediction and confidence interval shows that our MLE and latent predictions are appropriate.