Optimal Parameter Estimation and Prediction in a Poisson Time Series Model Using Monte Carlo Maximum Likelihood and Newton's Method

Yingbo Tong

2024-08-01

Introduction In this project, a Poisson time series model with latent variales was fit to count data, a common task in the epidemiology of non-infectious diseases. The model consists of a log-linear relation with latent variables controlling the trend over time. Given the data set $y = (y_1, ..., y_n)^T$, we aim to estimate parameters β_0 and σ^2, and predict the latent variables $u = (u_1, ..., u_n)^T$. Using all the predicted values, a point estimation of the poisson dataset can be generated, along with a confidence interval for said estimate.

Methods and Computational Strategy To estimate the parameters β_0 and σ^2, the project employed Monte Carlo Maximum Likelihood Estimation (MCML). This method involves simulating multiple scenarios of latent variables u using Monte Carlo techniques, which then inform the likelihood estimation of observed data given these latent variables.

The log-likelihood function was approximated using these simulations, averaging over multiple Monte Carlo samples to stabilize the estimation against the variability inherent in such stochastic processes. The specific likelihood function used was:

$$L(\beta_0, \sigma^2) \approx \log\left(\frac{1}{L}\sum_{l=1}^{L} f(y|\sigma z_l)\right)$$

where $L = 10000$ is the number of Monte Carlo samples, and f denotes the model's likelihood function based on the Poisson distribution with parameters transformed by the latent variables z, with the distribution

$$z_l \sim N(\mathbf{0}, Q^{-1}),$$

with $Q = D^T D$ and D being the differencing matrix. Newton's algorithm was applied to optimize the Monte Carlo-approximated log-likelihood function, with initial parameter guesses of $\beta_0 = 0$ and $\sigma = 0.1$. The algorithm employs step-halving to restrict the step size to be always ≤ 2, as well as taking the absolute value of the hessian matrix for proper reflection. The algorithm also uses estimates of the gradient and the hessian given by the R library numDeriv to skip the derivation process, but came at a cost of increased running time. This optimization process iteratively updated the parameter estimates, and we were able to converge to a value of

$$\hat{\beta}_0 = 0.22443, \ \hat{\sigma} = 0.08327.$$

To calculate the confidence intervals for these estimates, we will obtain the estimate for the covariance matrix as

$$\text{Cov}(Y|u) \approx I^{-1}(\hat{\beta}_0, \hat{\sigma})$$

where

$$I(\theta) = \mathrm{E}\left(\frac{\partial^2 \ell}{\partial \theta^2}\right)$$

which means that an estimate for the covariance matrix is the inverse of the parameter estimate for the negative log likelihood hessian. We calculate the covariance matrix and store it normally since it's only a 2×2 matrix. We then take the square root of the diagonals as the standard error for our parameters. Finally, we get that the confidence interval for those parameters are

$$\beta_0 : (0.1702, 0.2787) \text{ and } \sigma : (0.0583, 0.1082)$$

Then the final estimate and confidence interval for σ^2 would be

$$\hat{\sigma}^2 = (\hat{\sigma})^2 = 0.006934$$

And using the fact that the CI for $\hat{\sigma}^2$ is $(\hat{\sigma}^2 \pm 2|\hat{\sigma}|\mathrm{se}(\hat{\sigma}))$, we get

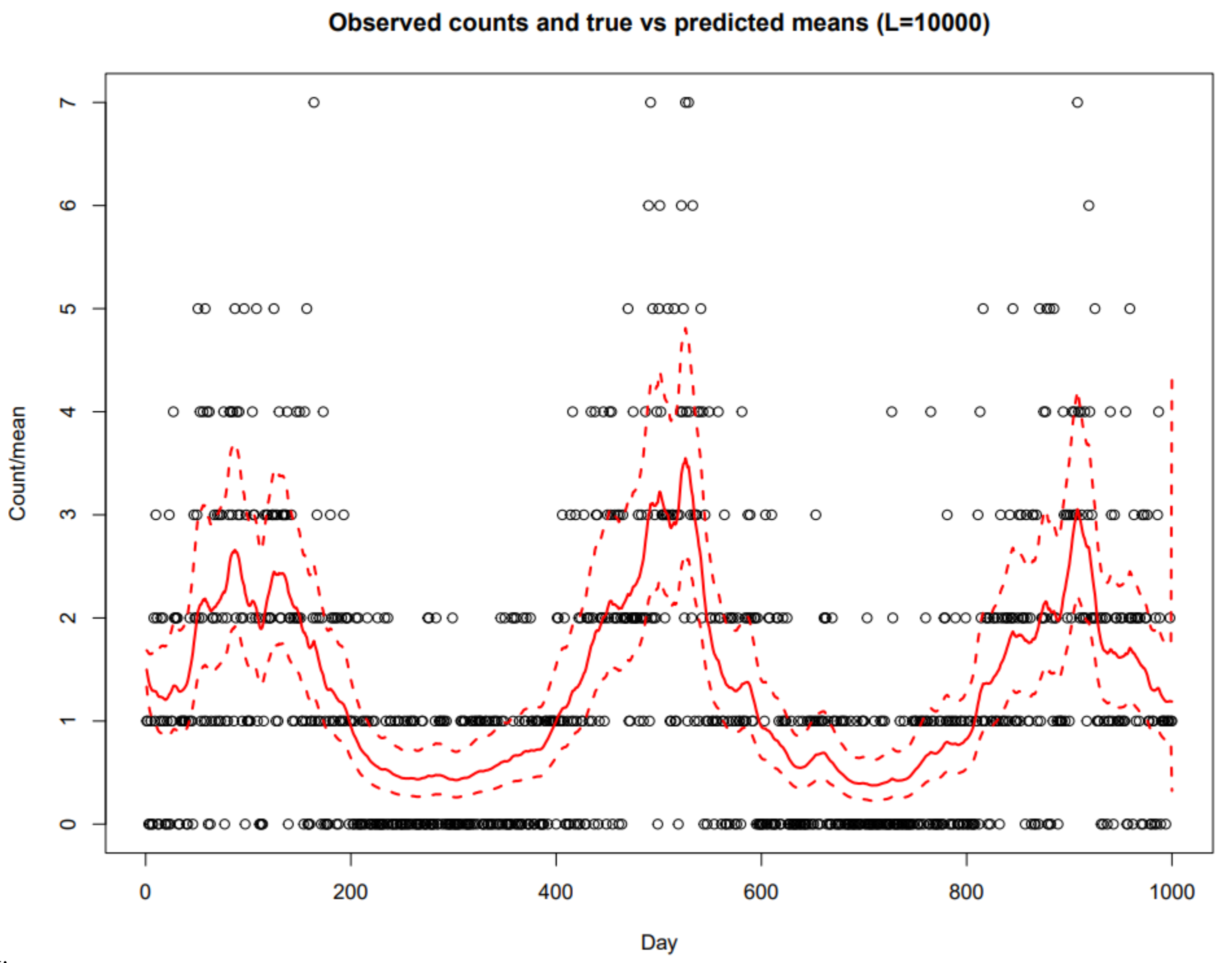
$$\hat{\sigma}^2 : (0.00286, 0.01101)$$

We obtain values for our standard errors that are quite small. This is most likely due to having more than enough data points from our sampling of y, as well as having large amounts of sampling ($L = 10000$) from our Monte Carlo simulations.

After the parameter estimation, the latent variables u were then also estimated using a similar maximum likelihood approach specific to the Poisson model. Given the estimates $\hat{\beta}_0$ and $\hat{\sigma}^2$, the distribution of u conditioned on y was approximated using:

$$\hat{u} = \arg\max_u P(u; y, \hat{\beta}_0, \hat{\sigma}^2)$$

This process involved using Newton's method again, initialized with the entire vector u set to $\hat{\beta}_0$. The iterative updates provided estimates for u, and is then illustrated with the plot

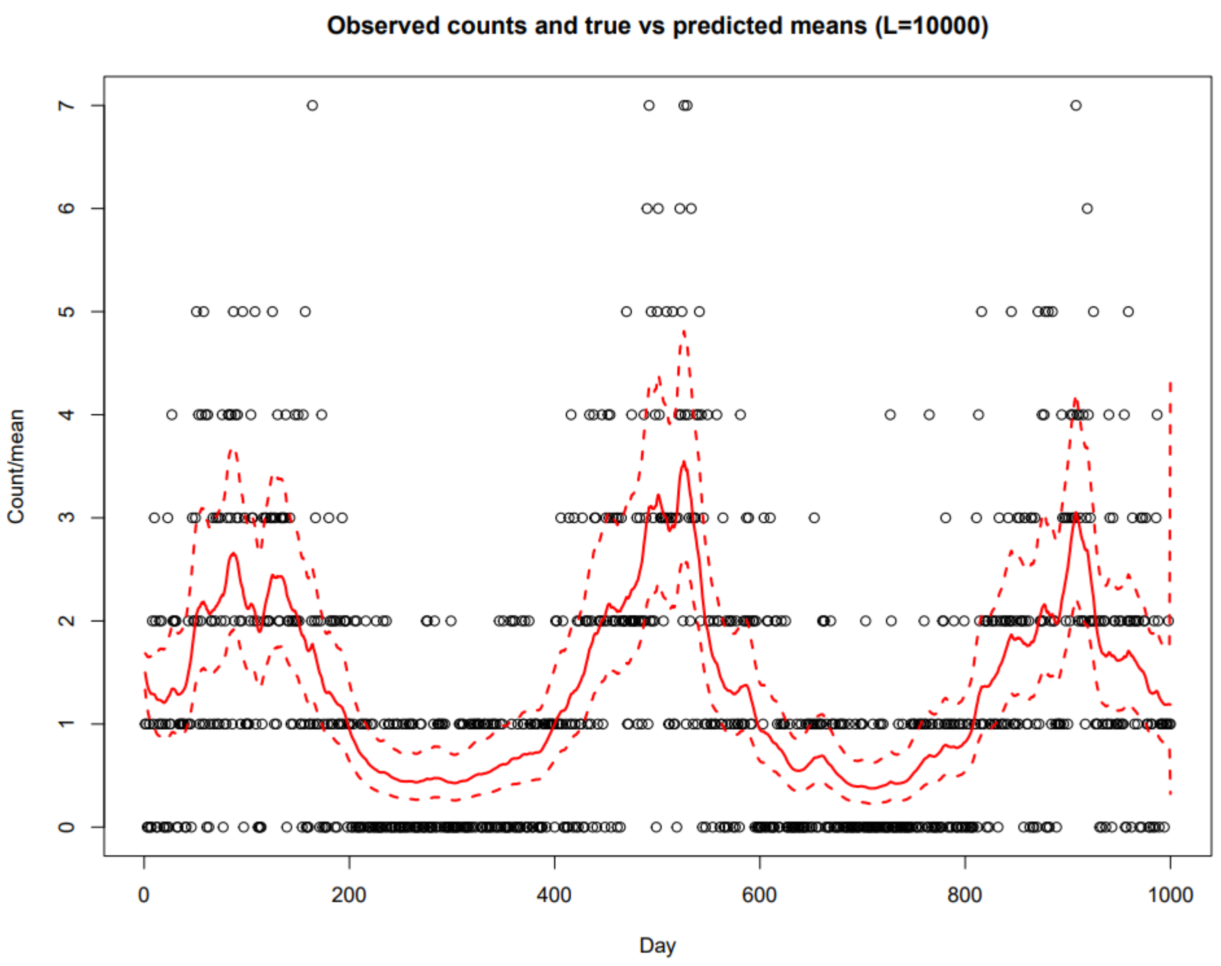


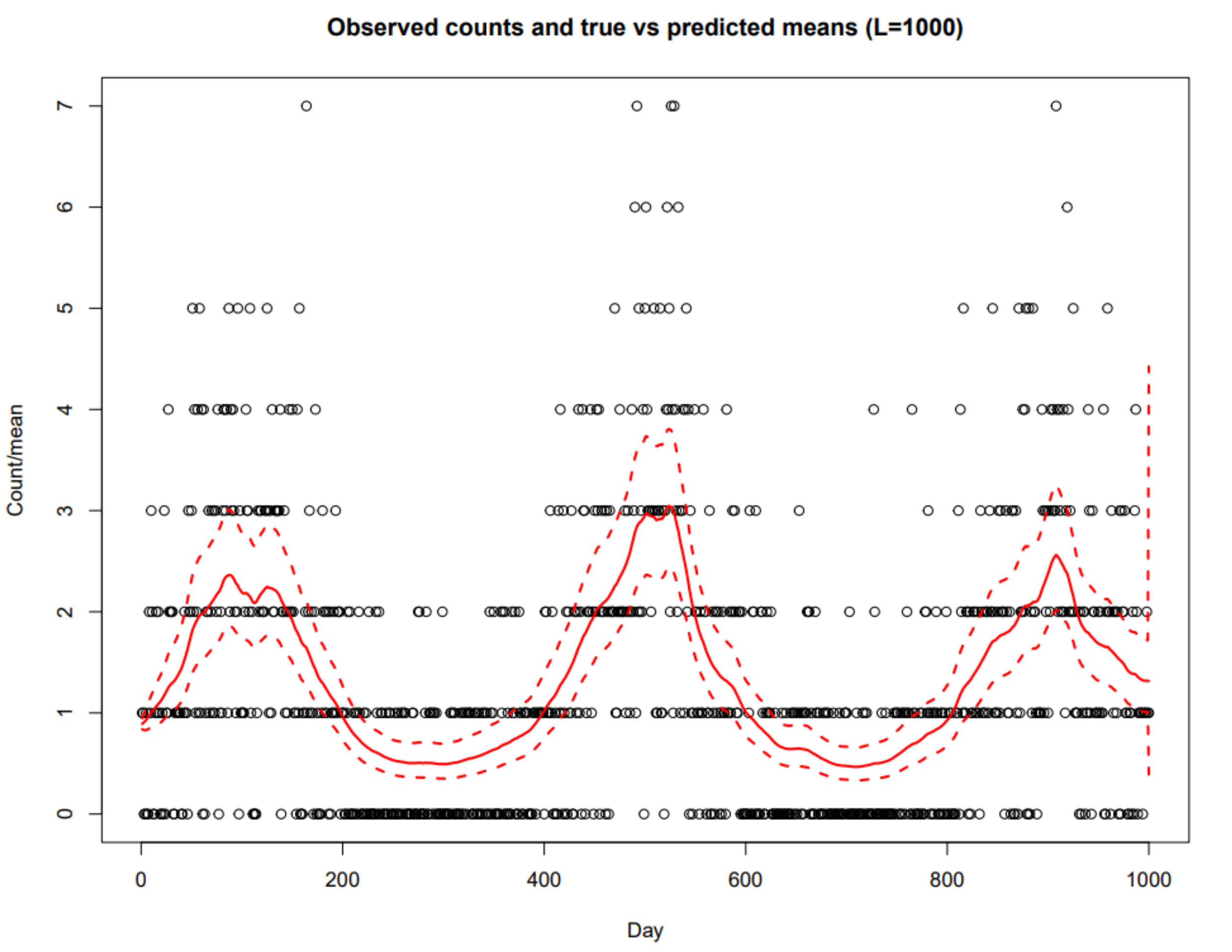
below:

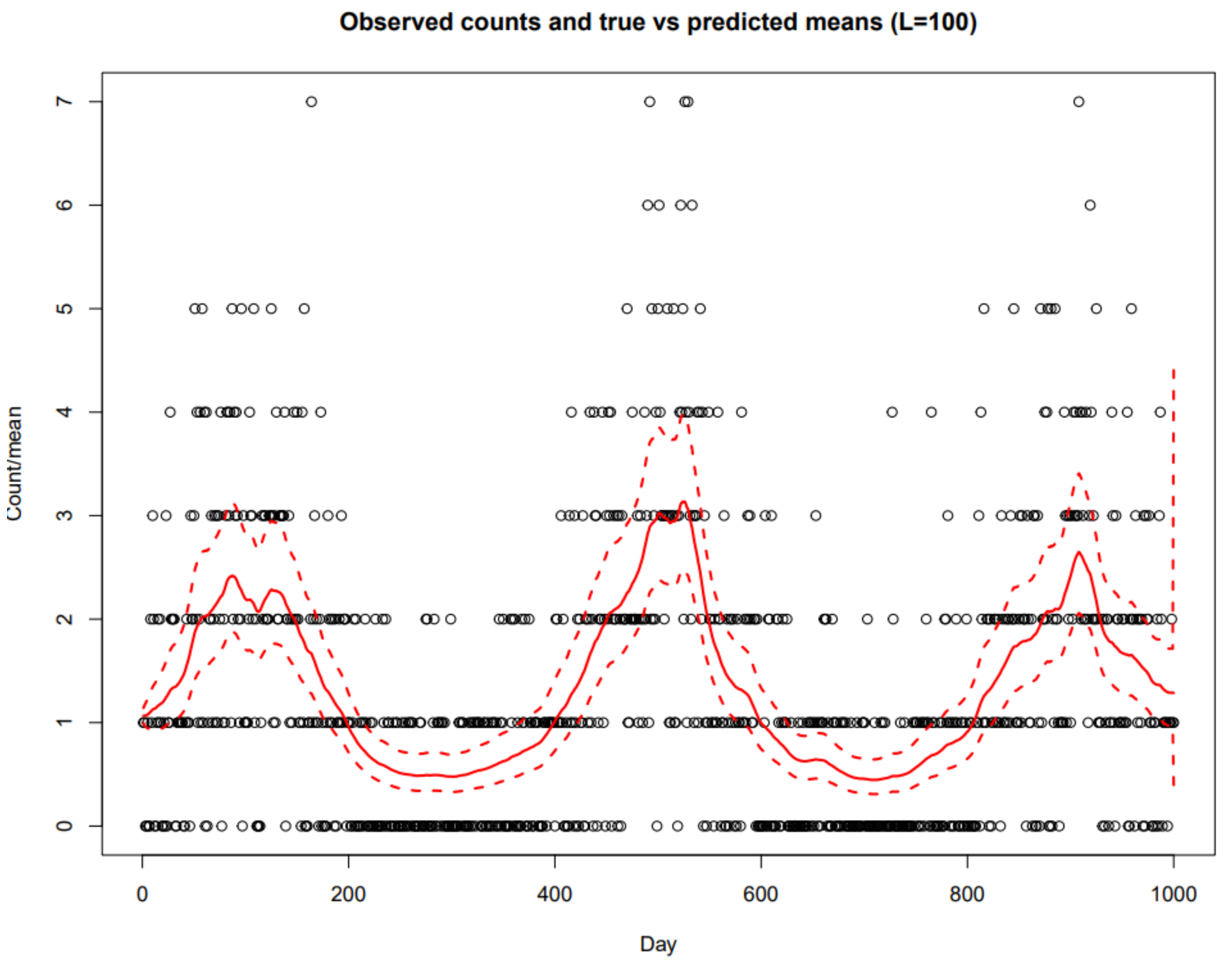
The red line in the plot shows the estimated value for $\hat{\lambda}_i = \exp(\hat{\beta}_0 + \hat{u}_i)$. Note that the plot of the estimates captures the trend very well, and includes a reasonable amount of noise without losing the general trend.

Choices of Parameters and Starting values Initially, in order to use the MCML method to optimize for the estimates of β_0, σ, a starting value is needed. To choose this value, we decided to choose a range of starting values of $\beta_0 : (-2, 2)$ and $\sigma : (0, 0.5)$ and performed Newton's method to find the estimate. All value pairs within those starting ranges ended up converging to the same points. Finally, $(0, 0.1)$ was chosen as the starting point, Further, running the `optim` function in R generated the same estimations, which solidifies that our choice of starting values most likely led to the global minimum. The starting values for the second optimization for estimating $\mathbf{u}$ were simply chosen as $(\hat{\beta}_0, \ldots, \hat{\beta}_0)$ as per the start code file.

The choice of our L depends on a balance of accuracy and efficiency. Having a larger value of L will increase the accuracy of our MCML estimations, but will also in turn put a toll on our computation, thus making the running time of the code lengthy. We compiled the following images for visualizing the impacts of different sizes of L:


Observed counts and true vs predicted means (L=10000)
Count/mean
Day
0
1
2
3
4
5
6
7
0
200
400
600
800
1000


Observed counts and true vs predicted means (L=1000)
Count/mean
Day
0
1
2
3
4
5
6
7
0
200
400
600
800
1000



As L increases, the patterns within the estimates of the points become a closer fit for our data points.

Computational Strategies To utilize MCML and estimate $\hat{\beta}_0, \hat{\sigma}$, the probability distribution function $f(y|u;\sigma,\beta_0)$ must be implemented. The way we handled the product of 1000 values between 0 and 1 without underflowing to 0 was using the following `logSumExp` trick:

$$\hat{\ell}(\beta_0,\sigma^2;y) = \log \int f(y|u;\sigma,\beta_0) f(u;\sigma^2) du$$

$$\approx \log \left(\frac{1}{L} \sum_{l=1}^{L} \exp \left[\sum_{i=1}^{n} (y_i(\sigma z_l + \beta_0) - \exp(\sigma z_l + \beta_0) - \log y_i!) \right] \right)$$

$$\approx \text{LogSumExp} \left(\sum_{i=1}^{n} (y_i(\sigma z_l + \beta_0) - \exp(\sigma z_l + \beta_0) - \log y_i!) \right) - \log L$$

Which is a stable and reliable way to estimate the log likelihood function that retains the minimum at the same place as the non estimated function.

There were operations requiring matrices of $n \times n$ dimension, namely the matrix Q, which was used as the inverse of the covariance matrix for generating samples of z_l, and the hessian matrix for the estimates of $\mathbf{u}$, H. Both matrices were stored as a sparse matrix using functions in R, as well as using the Cholesky decomposition whenever we need to use the matrices for calculation. No matrices of size $n \times n$ were ever stored or directly multiplied or inverted during any process of this statistical procedure, and thus our time and space complexity is $<< O(n^3)$, which is a huge increase in speed performance.